12-2019

Improving Manufacturing Data Quality with Data Fusion and Advanced Algorithms for Improved Total Data Quality Management

David Juriga
University of Tennessee, djuriga@utk.edu

To the Graduate Council:

I am submitting herewith a thesis written by David Juriga entitled "Improving Manufacturing Data Quality with Data Fusion and Advanced Algorithms for Improved Total Data Quality Management." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Forestry.

Timothy Young, Major Professor

We have read this thesis and recommend its acceptance:

Alexander Petutschnig, Bogdan Bichescu, Terry Liles

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Improving Manufacturing Data Quality with Data Fusion and Advanced Algorithms for Improved Total Data Quality Management

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

David Christoph Juriga

December 2019

Acknowledgements

I want to express my sincere gratitude to Dr. Timothy M. Young for his guidance, support, and encouragement during my studies as well as allowing me to pursue the opportunity to study at the University of Tennessee.

Furthermore, I would like to express my appreciation to Dr. Alexander Petutschnigg for his knowledge and support from the Salzburg University of Applied Sciences throughout the past year.

I thank my committee members Dr. Bogdan Bichescu and Dr. Terry Liles for their support and suggestions throughout my studies and creation of my thesis.

Special thanks to the coworkers from the Center for Renewable Carbon, Mr. Chris Helton, Mr. Anton Astner, and Mr. Qijun Zhang for your support and companionship. Thank you to Mr. Elliot DeVore, whose assistance and support during my studies have been invaluable.

Funding for this project was provided by the U.S. Department of Agriculture, National Institute of Food and Agriculture, Agriculture and Food Research Initiative, McIntire-Stennis Project (Young, TEN00MS-107), and the University Institute of Agriculture - Center for Renewable Carbon.

Last, I thank my friends and family for their incredible support throughout all my studies and growth over the past years.

Abstract

Data mining and predictive analytics in the sustainable-biomaterials industries is currently not feasible given the lack of organization and management of the database structures. The advent of artificial intelligence, data mining, robotics, etc., has become a standard for successful business endeavors and is known as the 'Fourth Industrial Revolution' or 'Industry 4.0' in Europe. Data quality improvement through real-time multi-layer data fusion across interconnected networks and statistical quality assessment may improve the usefulness of databases maintained by these industries. Relational databases with a high degree of quality may be the gateway for predictive modeling and enhanced business analytics.

Data quality is a key issue in the sustainable bio-materials industry. Untreated data from multiple databases (*e.g.,* sensor data and destructive test data) are generally not in the right structure to perform advanced analytics. Some inherent problems of data from sensors that are stored in data warehouses at millisecond intervals include missing values, duplicate records, sensor failure data (data out of feasible range), outliers, etc.. These inherent problems of the untreated data represent information loss and mute predictive analytics. This data science focused research was to create a continuous real-time software algorithm for data cleaning that automatically aligns, fuses, and assesses data quality for missing fields and potential outliers. The program automatically reduces the variable size, imputes missing values, and predicts the destructive test data for every record in a database. Improved data quality was assessed using 10-fold cross-validation and the normalized root mean square error of prediction statistic.

The impact of outliers and missing data were tested on a simulated dataset with 201 variations of outlier percentages ranging from 0-50% and missing data percentages ranging from 0-50%. The software program was also validated on a real dataset from the wood composites industry. The number of sensors needed for accurate predictions are highly dependent on the correlation between independent variables and dependent variables. Overall, the data cleaning

software program significantly decreased the NRMSEP ranging from 64% to 12% of quality control variables for key destructive test values.

Table of Contents

List of Tables

List of Figures

CHAPTER ONE
INTRODUCTION AND GENERAL INFORMATION

Problem Identification and Explanation

Predictive analytics, data mining, and the use of 'big data' are paramount to success in international business and research communities. Data mining and big data are fundamental to 'Cloud Manufacturing' in the fourth industrial revolution known as 'Industry 4.0.', where computers and automation come together in a new way through remote connectivity to computer systems equipped with machine learning algorithms that are predictive (Zhong *et al.*, 2017). In this thesis, the sustainable biomaterials industries and related agricultural industries are defined as companies that manufacture: biofuels, bioenergy, lignocellulose products, Nano-biomaterials, wood composites (*e.g.,* particleboard, medium-density fiberboard), engineered wood (*e.g.,* oriented strand board, laminated veneer lumber, cross-laminated lumber), paper and paper products, processing of agricultural products (*e.g.,* rice processing, soybean processing, wheat processing, corn processing) from renewable feedstock sources. The Industries exist in highly competitive markets that are commodity-based, where competitive advantage is sought by lowering the final costs of the manufactured product. These Industries are essential to the U.S. economy, (American Forest & Paper Association, 2018) but are facing growing competition from imported products and movement towards non-renewable petroleum-derived products, *e.g.,* PVC flooring, plastic moldings, petroleum fuels, petroleum carbon fibers. As of 2017, the sustainable biomaterials industries accounted for approximately five percent of the total U.S. manufacturing gross domestic product (GDP) and employed more than 937,000 people. The industry met a payroll of roughly $50 billion in 2017 and was seen in the top ten manufacturing sector employers in 45 states (U.S Chamber of Commerce, 2018). These companies can benefit from participating in 'Industry 4.0' where predictive analytics and real-time models can improve decision-making related to process outcomes and improve product quality while minimizing cost.

Successful data mining is not attainable without digital data integration and high-quality data, *i.e.,* data of high value. ‘Big data’ is defined relative to its context; however, the standard definition is as multiple terabytes or petabytes. Research suggests, “*big data is a subjective label attached to situations in which human and technical infrastructures are unable to keep pace with a company’s data needs*” (Gandomi and Haider, 2015). In the context of sustainable biomaterials industries and the related agricultural industries, ‘big data’ from industrial processes may be only hundreds of gigabytes or less than one terabyte. However, big data of any size, that is of poor quality, have little value for business improvement from applications of data mining and predictive analytics.

The importance of big data, data mining, and Artificial Intelligence (AI) is emerging quickly in the sustainable biomaterials industries with the advent of ‘Industry 4.0’. Industry 4.0 is exemplified by the Composite Panel Association (CPA) annual meeting in March 2018 for its members with a focus on data mining and AI for manufacturing, and the LIGNA (https://www.ligna.de/home), the largest international gathering of sustainable biomaterials industries focused on ‘Industry 4.0.’ Predictive analytics that are accurate in predicting failures can help manufacturers reduce scrap product and rework (*e.g.,* reduce remanufacture of off-grade or failing production runs). These losses do not account for additional energy losses and labor costs due to poor quality. Predictive analytics may also help manufacturers diagnose unknown sources of variation from the process (Deming and Shewhart, 1986). Process variation in weight, thickness, solvent applications, and drying, create significant costs for manufacturers in that variation influences operational targets. The more variation in a process, the higher the required operational targets for the critical inputs required to maintain specification of final manufactured product, which represent additional costs not accounted for in scrap and rework alone (Taguchi *et al.*, 2005).

There is an excess of literature on data mining and big data (Barton and Court, no date; Oxley and Thorsen, 2006; Schadt *et al.*, 2011; Brown, Chui and Manyika, 2011; Gelman, 2011; Dumbill, 2013; Zhong *et al.*, 2017; Cordeiro,

Deschamps and Pinheiro de Lima, 2017; Bibby and Dehe, 2018; Jabbar *et al.*, 2018; Peres and Fogliatto, 2018; Qi and Tao, 2018 and many others). However, a review of the literature suggests a gap in data science research related to 'automated data fusion' which incorporates automated algorithms for 'data quality assessment and improvement.' An extensive review (Liao *et al.*, 2017) was done of the literature related to 'Industry 4.0' and found the majority of published research was insufficient in addressing digital integration of data and a lack of affordable data mining software. Moreover, the analyses suggest a vast gap exists between 'Industry 4.0' laboratory experiments, comprising 95.1% of published research, and industrial applications with 4.9% of published research (Liao *et al.*, 2017).

The proposed research effort will reduce this gap by developing algorithms for data fusion and data quality improvement for the sustainable biomaterials and agricultural-based industries. Theorin *et al.* (2015) called data the "hidden asset in manufacturing" while Panetto and Molina (2008) highlighted the lack of utilization of data in manufacturing. Theorin *et al.* (2015) perceived that for future manufacturing systems to be competitive, "*…they need to make better use of plant data, ideally utilizing all data from the entire plant. Low-level data should be refined to real-time information for decision making, to facilitate competitiveness through informed and timely decisions."* Theorin *et al.* (2015) estimated that 85% of all manufacturing data are unstructured and not useful for rapid decision making in high throughput production facilities. The proposed research effort directly addresses the problem defined by Theorin *et al.* (2015).

Most sustainable biomaterials and related agricultural processing companies gather real-time data from process sensors across programmable logic controller (PLC) networks stored in real-time data warehouses. Operational personnel retrieve such data to monitor and assess the stability of the process. A parallel information stream is also typically maintained by destructive testing or quality control (QC) laboratories where critical product quality data are gathered periodically throughout the production cycle (*e.g.,* tensile strength, modulus, water

absorption, protein content, starch content). Due to the time required for destructive testing or QC assessments, the time gap in data retrieval from the laboratories to operations personnel may be several hours. In the sustainable biomaterials industries, large quantities of a product are produced on high throughput presses during this time gap. Predicting real-time quality attributes between the time gaps from periodic laboratory samples may be invaluable to operations personnel in avoiding the manufacture of a defective product, or off-grade product. Data mining may also help diagnose sources of unknown variation in the process. Cost savings can be significant if a corrective action on sources of variation occurs, which leads to variation reduction of key inputs such as weight. Lowering operating weight is challenging since this reduces board density, which in turn, lowers strength. However, through data mining and modeling, other variables in the process can be identified that positively impact strength resulting in an ability to lower density while maintaining strength. Lower density saves material costs because fewer feedstocks are needed to manufacture the composite. Less feedstock usage helps us to more efficiently utilize our resource while lighter weight saves on transportation costs.

However, the vast majority of manufacturers in the sustainable biomaterials industries and agricultural processing industries have not integrated the destructive testing and QC laboratories databases with the real-time process data warehouses. The Industries typically have the destructive testing and QC laboratories databases on protected 'business networks' and PLC data warehouses on the protected 'process networks.' Integrating large amounts of digital data stored in multiple databases across networks by data fusion with data of high value is the gateway for discovery using 'Cloud Manufacturing' and advanced data mining with real-time predictive analytics.

The Importance of Data Quality (DQ). -- This thesis is aligned with research focused on improving the data quality of integrated databases which is the essential platform for data mining. Even though there is a plethora of literature on DQ, the following citations and quotes are the motivational theme for this thesis.

Francisco *et al.* (2017) and Ardagna *et al.* (2018) defined data quality (DQ) as an important issue for modern organizations, mainly for decision-making based on information. Francisco *et al.* (2017) indicated "…*that in order to obtain quality data, it is necessary to implement methods, processes, and specific techniques that handle information as a product with well established, controlled, and managed production processes, e.g., TQDM – Total Quality Data Management.*" Ardagna *et al.* (2018) stated: "…*data can create a real value only if combined with quality: good decisions and actions are the results of correct, reliable and complete data, i.e., methods and techniques for the data quality assessment can support the identification of suitable data to process*."

Cai and Zhu (2015) further discuss, "*high-quality data are the precondition for analyzing and using big data and for guaranteeing the value of the data,"* also see Batini *et al.* (2015) and Dumbill (2013). Gupta and Rani (2018) highlight the challenges in terms of "*data capture, storage, manipulation, management, analysis, and the wide gap that exists between big data potential and realization*" given many data quality shortcomings. Gupta and Rani (2018) indicated the need for research efforts in academia to assist Industry in the understanding of big data and data quality. Liu *et al.* (2016) further articulate this point, "*big data brings lots of 'big errors' in data quality and data usage...information incompleteness is one of these problems.*"

This thesis addresses the problem of information incompleteness (or information loss) and the development of automated algorithms for data quality improvement to support the advancement of data science research as applied to improve the competitiveness of the sustainable biomaterials industries. The research, on 'data quality science' to advance 'Industry 4.0' for the sustainable biomaterials industry and agricultural processing industries is not well documented in the public domain literature.

Rationale for Thesis

The rationale for the research is that data mining, big data, and predictive analytics are driving change in the 4th industrial revolution. Data mining and the use of big data are almost a prerequisite for sustaining a successful business venture. Fundamental to successful data mining is developing and maintaining high quality integrated digital data. The sustainable biomaterials industries and related agricultural industries are essential contributors to the U.S. economy, as highlighted in the economic statistics to follow in the next section. The sustainable biomaterials industries manufacture carbon-friendly products made from renewable natural resources. However, these industries exist in highly competitive markets due to increased international competition and product substitution with non-renewable, petroleum-derived products.

Given the low margins associated with commodity production, manufacturers in the sustainable biomaterials industries and related agricultural industries are highly competitive. Competitiveness prohibits sharing methods for data fusion, data quality assessment, or predictive modeling and slows down research in this area.

There is a gap in the literature in developing algorithms for data fusion and assessment of data quality on an industrial scale, *i.e.,* an interactive data depository for data fusion and data quality improvement. In this research, automated algorithms were developed to fuse multi-layer digital databases. After data fusion, automated algorithms were developed to assess and improve the quality of the data. This thesis attempts to make a marginal contribution to narrow the enormous gap between academic research focused on data science and useful applications in Industry, *i.e.,* recall Liao *et al.* (2017) stated this gap to be as large as 90%.

Objectives

The objectives of this thesis are:

1) Develop software that will allow automated digital data integration and data fusion of two databases;
2) Develop algorithms for data quality assessment;
3) Develop a more advanced knowledge of the impact of outliers, missing data, quantity of independent variables, and quantity of dependent records on the database and the predicted values on simulated databases;
4) Validate the software system on a real dataset.

CHAPTER TWO
LITERATURE REVIEW

Industrial Revolution and Industry 4.0

In the last century, technological developments, lower costs and barriers linked to data collection and storage in industrial environments (Arcidiacono and Pieroni, 2018; Peres and Fogliatto, 2018) have led to three industrial revolutions. The advent of artificial intelligence, data mining, robotics, etc., as a standard for successful business endeavors, has resulted in a 'Fourth Industrial Revolution' known in Europe as 'Industry 4.0.'. The 'Fourth Revolution' started in early 1970 with the progression of electronics and information technologies towards a higher level of automation in manufacturing facilities (Figure 1) (Stock and Seliger, 2016). In 2011, the Germans coined 'Industry 4.0' with a proposal for "*development of a new concept of German economic policy based on high-tech strategies*" (Roblek, Meško and Krapež, 2016; Cordeiro, Deschamps and Pinheiro de Lima, 2017). Industry 4.0 spread first in Germany followed by the European Union (EU), North America, Asia, and the rest of the world (Liao *et al.*, 2017). In terms of published literature, Europe has 83% of all publications on the subject, followed by Asia with 13.8% and 3.2% from the other continents (Liao *et al.*, 2017).

The fourth industrial revolution has many definitions. It is the change from internet and intranet to the 'Internet of Things,' from lean production to smart factories and from automation and computerization to visualization and integration (Roblek, Meško and Krapež, 2016). Based on connectivity, supply chain integration and Cyber-Physical Systems are capable of the inclusion and adoption of new applications and technologies (Cordeiro, Deschamps and Pinheiro de Lima, 2017). Industry 4.0 includes concepts such as the 'Internet of Things,' 'Internet of Services,' 'Cyber-Physical Systems,' which communicate via the internet with the customer to customer and customer to machine exchange (Roblek, Meško and Krapež, 2016).

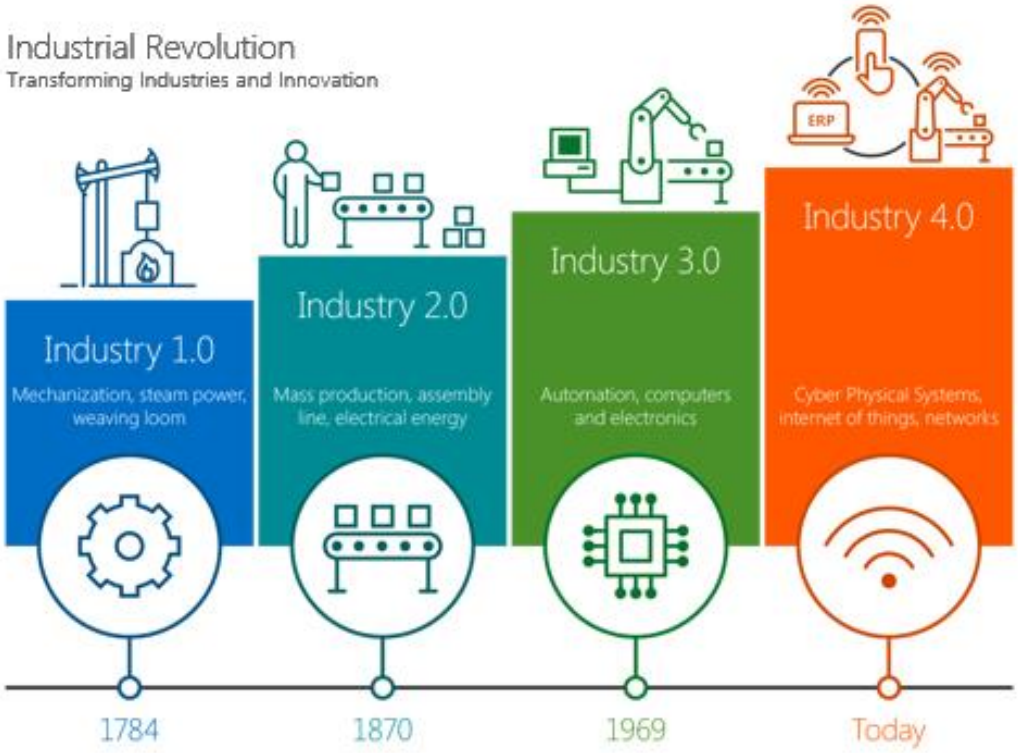

Figure 1: Industrial revolution: transforming industries and innovation (Morgan, 2018).

The fourth industrial revolution started in Germany as Industry 4.0, in the EU as Factories of the Future and in the U.S. as Industrial Internet (Stock and Seliger, 2016; Liao *et al.*, 2017). Now the majority of the papers uses the term Industry 4.0.

Many companies invest in Industry 4.0 including ABB, AT&T, Bosch, Cisco Emerson Electric, General Electric, Hitachi, Honeywell, IBM, Intel, Mitsubishi Electric, Panasonic, Schneider Electric, Siemens (Liao *et al.*, 2017). The justification given is for supply chain and logistics improvement. Industry 4.0 can improve logistics and supply chain efficiency by providing information from big data that is 'real-time' and more detailed than current systems. Efficiency improvements mitigate such things as the bullwhip effect (increased inventory swings further in the supply chain, due to shifts in customer demand), reduce counterfeiting and improves product traceability (Roblek, Meško and Krapež, 2016). Other sectors adopting Industry 4.0 include smart infrastructure, health care, and security and privacy (Roblek, Meško and Krapež, 2016).

The main factors of Industry 4.0 are digitalization, automation and linking (Roblek, Meško and Krapež, 2016). Digitalization of the whole production is the first step in automating systems for improved production from machines (Roblek, Meško and Krapež, 2016). There are four key components to achieve the main factors of Industry 4.0 (digitalization, automation, and linking). The components are: 1) Cyber-Physical Systems (CPS); 2) Internet of Things (IoT); 3) Internet of Service (IoS); and 4) the Smart Factory. (Roblek, Meško, and Krapež, 2016; Mrugalska and Wyrwicka, 2017).

A CPS is a decentralized mechanism algorithm supervised and connected through and to the internet and its users. The CPS also includes smart products and products with sensors and microchips to communicate (Roblek, Meško and Krapež, 2016; Stock and Seliger, 2016; Mrugalska and Wyrwicka, 2017). The IoT is the interconnection of devices via the Internet of Devices embedded in everyday objects. Mostly this information exchange is machine to machine (Roblek, Meško and Krapež, 2016; Stock and Seliger, 2016). The Internet of Services is the

interconnection of Services to customers (*e.g.,* social media) (Roblek, Meško and Krapež, 2016). The term Smart Factory applies to the factory, which will be more efficient, dynamic, and flexible. Smart Factory equips manufacturing with sensors, actors and autonomous systems. Machines and equipment will have the ability to make decisions and to improve themselves via algorithms and machine learning (Roblek, Meško and Krapež, 2016; Mrugalska and Wyrwicka, 2017). The Sustainable Biomaterials Industry is in the researching phase to develop methods to get from Industry 3.0 to Industry 4.0.

Sustainable Biomaterials Industry

As of 2017, the sustainable biomaterials industries accounted for approximately five percent of the total U.S. manufacturing gross domestic product (GDP) and employed more than 937,000 people. The industry met a payroll of roughly $50 billion in 2017 and was seen in the top ten manufacturing sector employers in 45 states (U.S Chamber of Commerce, 2018). Five percent of manufacturing GDP in the United States in 2017 was comprised of industry shipments of approximately $282 billion from a total of 5,292 manufacturing facilities that contributed state and local taxes for roughly one billion dollars in state and local taxes (U.S. Department of Energy, 2018). A study by the U.S. Department of Agriculture considered it an employment growth industry with an increase in demand for employment by more than five percent (57,900 openings) between 2015 and 2020 for college graduates with bachelors or higher degrees (U.S Chamber of Commerce, 2018). The agricultural processing industries contributed $82 billion in output or 5.5% of GDP in the U.S. in 2017 (Dunham and Associates, 2017). This Industry employed approximately three million Americans in 2017 (Dunham and Associates, 2017).

The U.S. Environmental Protection Agency (2017) noted, sustainable biomaterials industries are based on renewable and sustainable raw materials, *i.e.,* materials retrieved from forests and residues take carbon dioxide from the atmosphere and help the global carbon balance. However, traditional competitors,

such as Canada, Scandinavia, and Japan followed the global leader (US) for decades, and now emerging countries (Brazil, Chile, China, Indonesia) have entered in the renewable Industry. Technical challenges confronting the industry are deciding on using recycled materials cost-effectively, meeting environmental regulations, decreasing energy costs, and a declining land base (U.S. Department of Energy, 2018). The proposed research on improving the overall data quality for advanced data mining will assist this industry in providing new solutions for problems by using analytical-based research.

Big Data

Big data is defined as linkable information with large volumes and complete data structures (Khoury and Ioannidis, 2014; Liu *et al.*, 2016) that cannot be efficiently processed by standard database methods and tools (Batini *et al.*, 2015). Big data started as a technical problem; now, it is key to business success. In general, Laney (2001) Gandomi and Haider (2015) and Saidulu (2017) describe the three ‘V's’ of big data (Figure 2). The three ‘V’s’ are defined as 1) Volume (the volume of the big data should be large); 2) Variety (big data comes from different sources); and 3) Velocity (big data is updating in real-time or almost real-time). For manufacturing data, this means sensors frequently measure data points on a production line and store the data is an industrial data warehouse database, which addresses the first ‘V’ of volume. Measurement data are not just from sensors; quality control data are also measured in a destructive testing lab and stored in an independent database, providing the second ‘V’ of variety. The sensor data are measured in a frequency of milliseconds and are stored in real-time data, which addresses the third ‘V’ of velocity. Two more V’s were recently added to the original three ‘V’s’: veracity (to look at the trustworthiness) and value (useful information that can be extracted) (Gandomi and Haider, 2015). According to the five criteria (volume, variety, velocity, veracity, and value), the manufacturing data from the sustainable biomaterials industry are considered big data.

In the past decade, scientists started to focus on big data because information became available through new technologies in vast amounts (Liu *et al.*, 2016). The richness and availability of data continue to increase steadily (Liu et al., 2016). However, data scientists and manufacturers research the continuous improvement of data quality and data analytics and encounter significant problems. Practitioners sometimes tend to trust the integrity of big data sets without further analysis. Anderson (2008) wrote, "*with enough data, the numbers speak for themselves*." In contrary, Liu *et al.* (2016) stated that most of the big data datasets do not get supervised by scientific investigators or governmental agencies, which implies low authenticity and credibility of the results.

Big data sometimes comes with big problems. The first possible bias in the data sets is that companies collect data with the purpose of profit. Therefore, scientific methods like random and reliable sampling are sometimes violated (Liu *et al.*, 2016). Many companies have a free hand over the sampling methods and algorithms used. If the company decides to change its methods or algorithms of sampling, this change may not be easily detected in a continuous dataset. Subsequently, analysis of a dataset from shifted sampling methods can lead to erroneous results and models that are not robust (Batini *et al.*, 2015; Liu *et al.*, 2016).

A second problem is information incompleteness and noise in big data. The initial use of the data in commercial use and the repurposing of datasets results presents many issues. For example, major problems include the presence of null-values in the data-base that resulted from sensors failing and the time periods during which they were replaced. The sensor's accuracy (and or functioning) and signal-lines can create noise in the data or add variation.

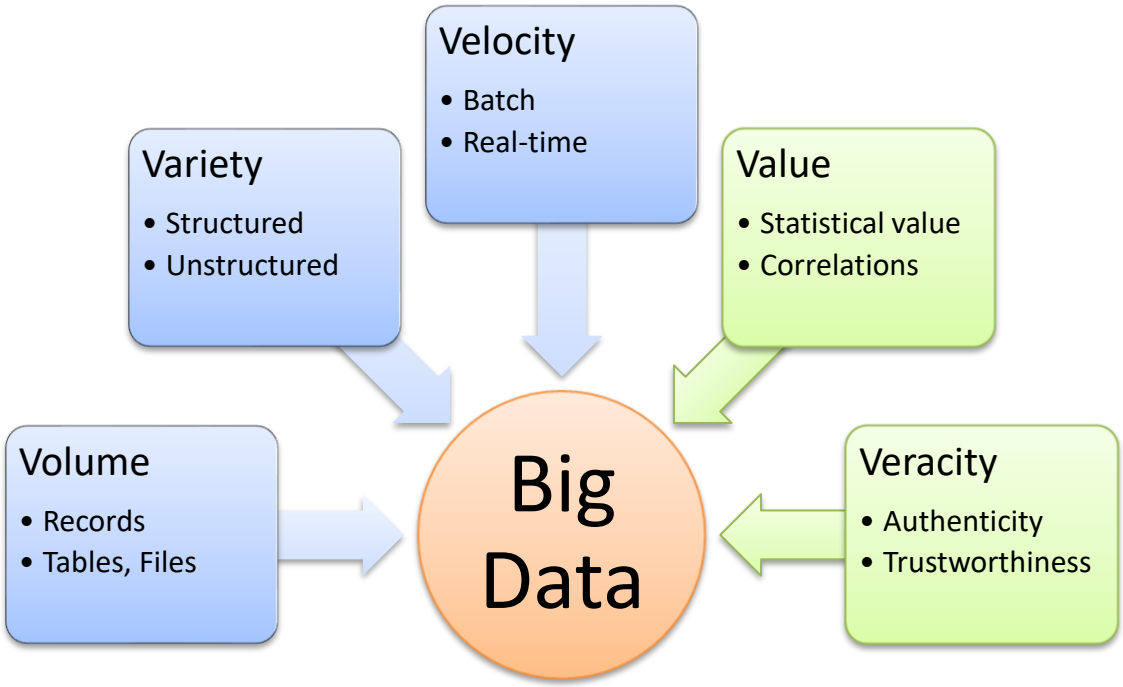

Figure 2: Five V's of Big Data, Volume, Variety, Velocity, Value, Veracity (Gandomi and Haider, 2015)

A third problem is the representativeness of big data. Businesses collect the data with the purpose of profit, which means the data represents the data of the company's interest (Liu *et al.*, 2016; Liu, Song and Duan, 2016; Cordeiro, Deschamps and Pinheiro de Lima, 2017). A fourth problem is not occurring in manufacturing big data, but the relatability and consistency of the analysis of data with algorithms in the background that incorrectly autocomplete sentences or inaccurately predict interests (*e.g.,* Google, Facebook or Twitter). Algorithms using big data can also lead to ethical issues in reporting of the results, *i.e.*, as soon as data from an individual can be tracked, the question arises on ethically acceptable and privacy issues (Liu *et al.*, 2016).

Big data in research should address big errors to find new reliable data analysis to decrease errors which are directly addressed in this thesis. Further, universities or public agencies should cooperate with data providers to adapt

research designs such as experimental designs to eliminate the influence of providers. Also, multiple data sources should be used to validate the findings based on big data. Research ethics and best practices should be worldwide and further implemented in governmental, university, and commercial usage (Liu *et al.*, 2016; Cordeiro, Deschamps and Pinheiro de Lima, 2017).

Programming Languages Fundamental to Thesis

According to Robinson (2017), Python® is a programming language predicted to dominate software applications in the near future (Figure 3). At the moment, however, Java® and C++® are considered by many to be better than Python® (Table 1). JavaScript® has more users currently, but the trend indicates that Python® has the highest increase in interest over time since 2012 (Table 1) (Petkov, 2018). Data analysis is the primary use of Python®, and web development is considered secondary. The performance of Python® is slower than C® or C++®, and the usage primary for Computers (Jodlowska, 2018). Open Source, supportive community, and a vast library are advantages of Python® (Robinson, 2017; Jodlowska, 2018). Python® is a general-purpose, interpreted, high-level programing language and was created in 1990 by Guido van Rosso (Python® Software Foundation, 2019).

Database and Database Systems

SQL® and NoSQL are the two dominant database systems. SQL® (Structured Query Language) is structured in tables with relations and follow the relational algebra rules. SQL® systems are suitable for data structured in rows, like manufacturing data. The database follows the on-write approach, where the datum must be inserted in a specific form to avoid errors. This approach also guarantees the same result for the same query when no new data are added (w3schools, 2019). NoSQL systems, like Hadoop® or MongoDB®, store data in unstructured form on several servers.

Table 1: Comparison of scripting languages (Petkov, 2018).

Language	TIOBE Rating [%]	GitHub pull [Million people]	usage trend
Java	14.21%	0.98	decrease
C++	5.60%	0.41	ups and downs
Python®	4.67%	1.00	impressive increase
JavaScript	3.47%	2.30	steady increase
Ruby	2.41%	0.87	decrease
R	N/A	N/A	steady rise

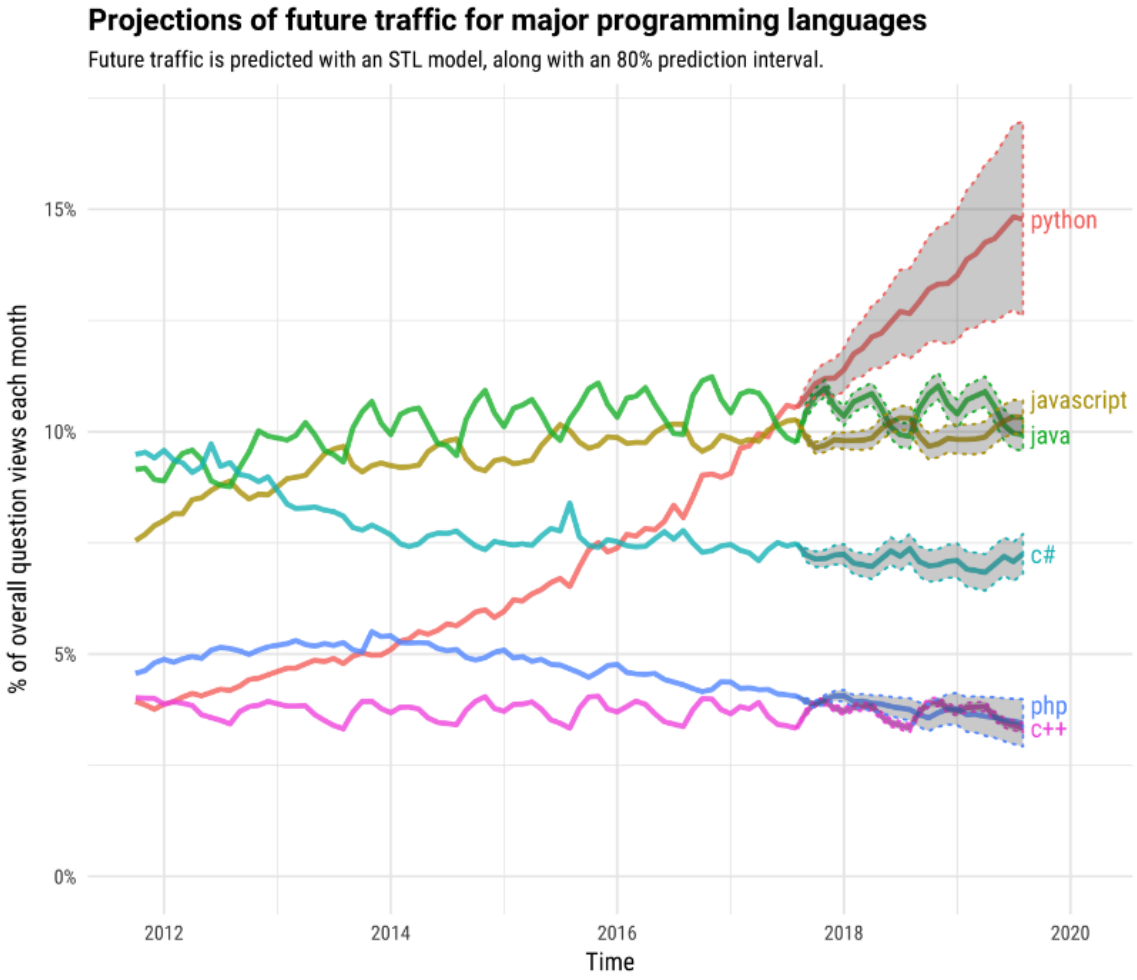

Figure 3: Projections of future traffic for major programming languages (Robinson, 2017)

The system is optimal for data whose form is unknown. Companies like Google or Facebook are using NoSQL systems for their inputs. NoSQL systems work with the on-read approach. It does not matter what format the incoming datum has; it gets stored. Requests in NoSQL systems search for keywords for a specific timespan and presents the found results. NoSQL queries do not have the same result every time (MongoDB Inc., 2019).

Data Fusion

Data fusion is a new paradigm for integrating data from multiple sources, where the fused dataset can generate more information than the sum of the single datasets (Iyengar, Sastry and Balakrishnan, 2003). The process of combining data and information from various sources is called data (Oxley and Thorsen, 2006). In the literature, data fusion and data integration often fall for the same procedure. Fusion is used to map various objects to a single object. Integration is used when smaller systems get connected to a bigger system (Oxley and Thorsen, 2006). Fused data helps companies reduce resources, increase competitiveness and support better decision foundations (Lund, 2017; U.S. Department of Transportation, 2017). Also, fields like data availability, accuracy, completeness, consistency, clarity, processing time, accountability can be improved with fused data (U.S. Department of Transportation, 2017). Data fusion can alleviate current problems and support the system (Iyengar, Sastry and Balakrishnan, 2003).

Variance, time-lag, and increasing data volume are problems for data fusion. Data collected from various sources vary in different ways (variance, outliers, missing values, number of products, frequency) (Qi and Tao, 2018), it includes natural variation, product variation, and different products. The number of installed sensors across the whole manufacturing value-chain is growing by a massive rate (Tao *et al.*, 2018). Next, the time-lag in manufacturing data causes biased or false conclusions in the analysis. The time-lag is the time difference between the measurements of one product on the production line, or as defined in the dictionary "*an interval of time between two related phenomena (such as a*

cause and its effect)" (Merriam-Webster, 2019). The technology for data collection and fusion, as well as data handling is not fully ready for smart manufacturing (Jabbar *et al.*, 2018; Tao *et al.*, 2018).

To fuse big data, the data link between the data tables must be determined (Augustian *et al.*, 2018). For process modeling the fusion relies on information gathered off- and online (Vandone, Baraldo and Valente, 2018). To create those links, there are three different data fusion architectures (Castanedo, 2013). The centralized architecture inputs the datum of the different sensors and aligns associates and estimates the datum to a complete master dataset (Figure 4). The opposite is the decentralized architecture (Figure 5) The fusion process generates multiple sub-datasets with parts of the aggregate data. This architecture can be useful when data cannot be directly transferred to a single product (Castanedo, 2013). The third architecture is the distributed architecture; it performs multiple centralized fusions and fuses those to the final fusion node. To be able to archive the data fusion, first, the software and devices must be integrated (Theorin *et al.*, 2015).

Increased efficiency is a huge factor in the papers on data fusion (Augustian *et al.*, 2018; Jabbar *et al.*, 2018; Qi and Tao, 2018; Tao *et al.*, 2018). The efficiency increase is argued by data collection (Tao *et al.*, 2018), quicker producibility of predictive models (Augustian *et al.*, 2018) and bottleneck (slowest/weakest performer in the system) detection (Qi and Tao, 2018). It is a step further to smart manufacturing (Tao *et al.*, 2018).

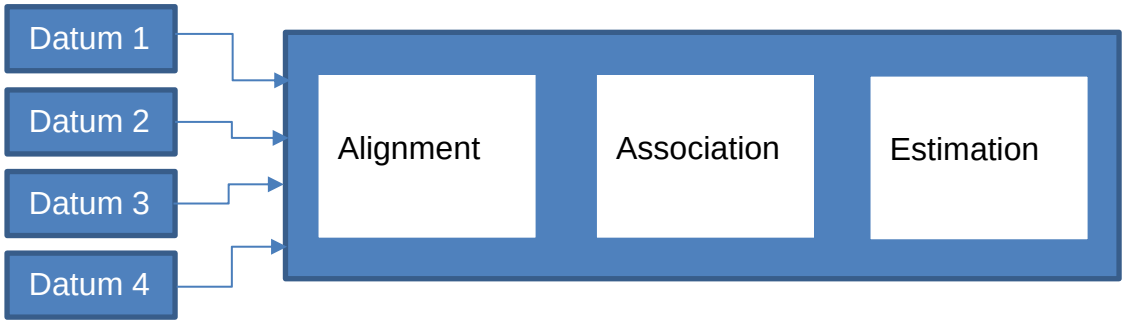

Figure 4: Centralized architecture of data fusion (Castanedo, 2013).

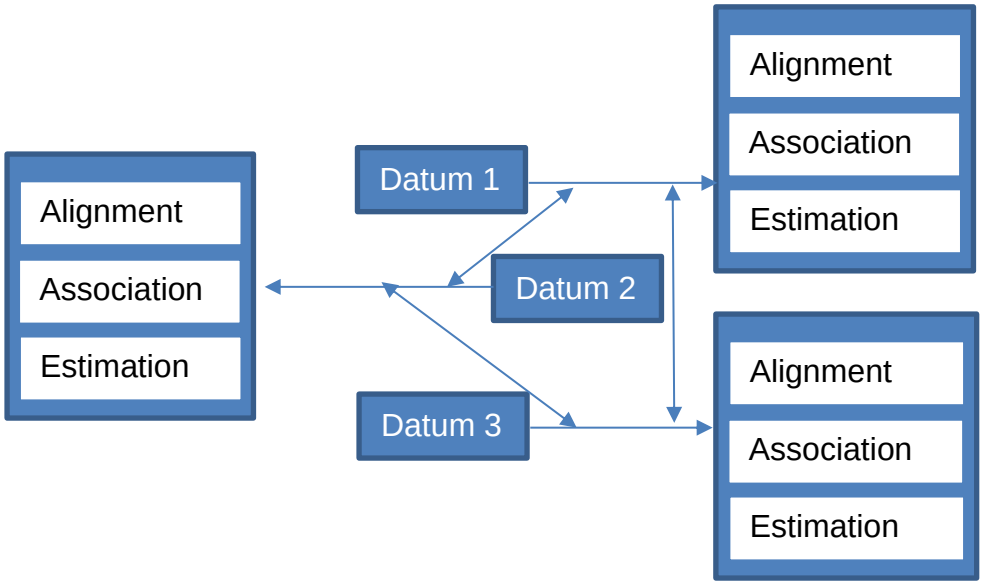

Figure 5: Decentralized architecture of data fusion (Castanedo, 2013).

Data Quality

"*Without big data analytics, companies are blind and deaf, wandering out onto the Web like deer on a freeway*" (Moore, 2012). The web made vast amounts of data as HTML documents available. HTML was designed to present data, not for computations (Rao, Gudivada, and Raghavan, 2015). Data warehouses were introduced in 1980 and brought up big data and big data quality issues. With the beginning of Industry 4.0, the focus on the quality of big data began (Rao, Gudivada, and Raghavan, 2015).

The range of applications for big data provides potential consequences for assessing data quality, which can have far-reaching impacts on analytical outcomes (Rao, Gudivada and Raghavan, 2015). 90% of the existing data is in an unstructured form (Saidulu, 2017). There is a shortage of awareness the need for high-quality data (Hazen *et al.*, 2017). According to Hazen *et al.* (2017) by implementing data quality methods, the data quality should improve; confidence in the quality of the data will expand, and the decision quality will be enhanced. 'Quality data' is essential for all analytical purposes (Becker, King and McMullen, 2015) and transcends across multiple information technology departments (Rao, Gudivada and Raghavan, 2015). Most data are often machine-generated by cameras, IoT networks, monitoring devices, social media or sensors (Rao, Gudivada and Raghavan, 2015) which makes future data management more complex, sensitive and combines them with potential errors (Thomas and Jacob, 2016).

Big data quality varies from one technology application to another. Correct and accurate instruments that generate raw data can be highly valuable for advanced analytics (Becker, King, and McMullen, 2015). With every measurement taken, the volume of the data increases, but the data quality is not ensured. Big data quality problems will generally increase proportionally (linear) to volume and variety of data collected (Becker, King and McMullen, 2015). Depending on the variety of sources and how homogenous or heterogeneous the data are, it provides varying levels of complexity to assess data quality (Becker, King and McMullen,

2015). Human-generated data quality issues are challenging to identify, understand, and correct (Becker, King and McMullen, 2015). Sensor data quality depends on the space, time and shape of data (Batini *et al.*, 2015) and is the focus of this thesis.

The term ‘data quality’ does not necessarily transcend to the term ‘big data quality.’ Data quality is the effort of collecting, assessing, and improving the quality of the data. ‘Big data quality’ implies further data algorithmic processing by collecting, fusing, analyzing quality, post-processing refinement, and the use of analytics of the data (Batini *et al.*, 2015).

Industry and research are exploring tools and techniques for managing big data quality (Becker, King and McMullen, 2015). An important step is that companies switch from human data entry to automated device capture, which improves the value of the data (Becker, King and McMullen, 2015). Current approaches to big data quality are rapidly evolving (Rao, Gudivada, and Raghavan, 2015). Data output and storage are part of almost every industrial process. Data warehousing and big data mining gives industries methods for robust, sustainable quality assessment and consistent processing (Micic *et al.*, 2018). However, developers must be sensitive with big data quality issues; *e.g.,* accurately identifying patterns of error in the data and discovering the root cause of such errors (Becker, Kin, and McMullen, 2015).

Quantitative and qualitative data assessments are used to determine the fitness for the usage of data. Researchers look at several factors to assess the data; Micic et al. (2018) and Batini et al. (2015) both describe the factors completeness, consistency, uniqueness, redundancy, accuracy, accessibility, and trust. After assessing the fit for use that gets modified to improve the data quality, process modeling, and predictive analysis is the central approach (Rao, Gudivada and Raghavan, 2015).

The predictive analysis brings a variety of techniques that are rooted in statistical significance. Small samples from the population should represent the whole relationship of variables, but small sample sizes do not represent the big

data (Gandomi and Haider, 2015). To get the right algorithm to improve quality, heterogeneity, noise, correlation, causation, and the independence of errors has to be analyzed (Gandomi and Haider, 2015).

Data Imputation

Missing data are omnipresent in big data. In the literature, the missing data are divided into three categories, missing at random (MAR), missing not at random (MNAR), and missing completely at random (MCAR) (Nakagawa and Freckleton, 2008).

MCAR means that variables are not correlated to missing values. MAR describes missing data in the response variable that is unrelated after adjusting one or more variables. MNAR explains missing values in the variable that is dependent on the variable (Nakagawa and Freckleton, 2008). MCAR, MAR, and MNAR are visualized in Figure 6. In the case of manufacturing data, the data should be MAR or MCAR; those can be justified in the presence of a rich set of predictors in the model (Allison, 2000). Analysis with complete data records and the exclusion of incomplete records can result in information loss or bias. Therefore scientists attempt to impute data to reduce this information loss.

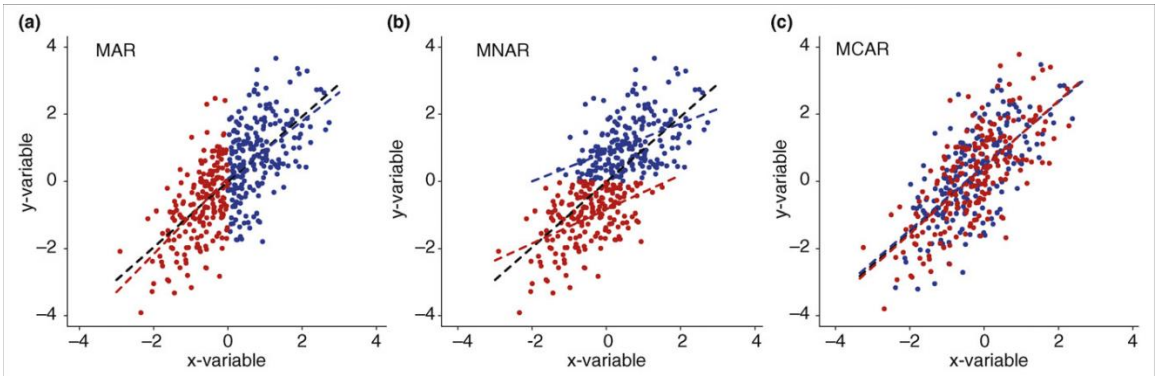

Figure 6: Illustrations of the three missing data categories, MAR, MNAR, MCAR. Blue represents missing values; red dots are measured values. (Nakagawa and Freckleton, 2008).

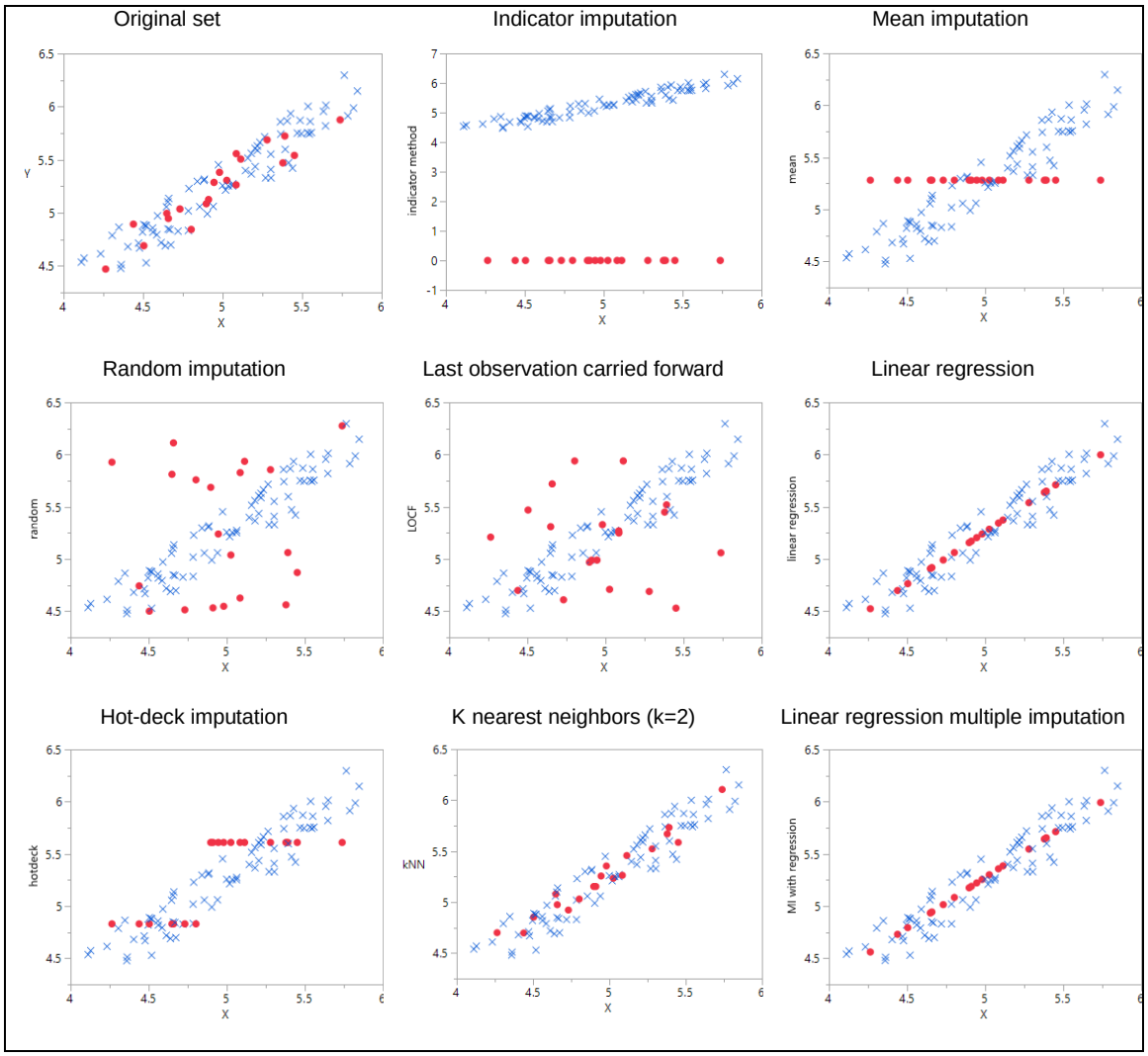

Figure 7: Visualization of eight different imputation methods. Blue values represent the test dataset (80% of the data), red values are the imputed data points (20% of the data).

Table 2: Comparison of different imputation methods.

Method	How	Y=
indicator method	an indicator replaces missing values	Y=0
mean/median	missing values replaced by mean/median	Y= Mean/Median
random	missing values replaced by random	Y= Random
LOCF	missing values replaced the last known value	$Y=Y_{i-1}$
hot deck	a median of a small subset replaces missing values	$Y=\text{mean}_{\text{subset(s)}}$
linear regression	missing values are replaced by regression on the predictor variable	Y= formula from regression
kNN	missing values replaced by the mean of the k nearest observations	$Y=\text{mean}_{\text{k nearest neighbors}}$
multiple imputation	missing values are replaced mean of several single imputation methods on a small sample	

There are several data imputation methods. The mean, mode, or median imputation is the fastest imputation method; it is efficient when a minimal amount of data is missing (Zhang, 2016). Mean imputation with lots of missing data causes a bias in the analysis (Figure 7) (Zhang, 2016). The last observation carried forward (LOCF), or last known value method imputes the last known value into the database, this might fix the problem with the current variable, but correlations to other variables are not provided with this imputation method, though the values can be close with autocorrelated data. The same problem occurs with random imputation (Figure 7). To recreate correlations between variables, the regression imputation was introduced (Allison, 2000). Simple linear regression or multiple linear regressions are performed to recreate the missing value. Regression imputation might produce more accurate values but is problematic if more than one value is missing (Zhang, 2016). An effective way to impute data is the k nearest neighbor method. The imputed value is the value of the closest neighbor or the median of the k nearest neighbors. The hot deck method uses one of the other methods, with the difference that the dataset gets split into logical subsets (decks)

(age groups, product types). The multiple imputation method splits the dataset into smaller random subsets and trains the method on multiple sets. Every training method calculates values for the primary dataset, and a pooled value gets imputed. Multiple imputation increases random error (Allison, 2000; Zhang, 2016). The main problem of these methods is that all are biasing the random error (Zhang, 2016).

Predictive Modeling

The procedure of predictive modeling uses data and statistics to predict future results with algorithms (Özel and Karpat, 2005; Kuhn and Johnson, 2013). A predictive model creates correlations between given explanatory variables and dependent variables (Gartner Inc., 2019; MicroStrategy, 2019). To create a successful predictive model, seven steps (Figure 8) are advised (Özel and Karpat, 2005; MathWorks Inc., 2019). Cleaning the data is the first step (removing NULL-values and false measurements and aligning the data). The second step is to identify if the modeling approach is parametric or nonparametric. The third step includes preprocessing of the data into a format suitable for a model. Next, the dataset gets split into training and validation data sets, followed by checking the goodness of fit of the model. Validation of the fit occurs on the validation data set. The last step is the use of the model to predict future data (Özel and Karpat, 2005; MathWorks Inc., 2019).

Predicted data are needed to make more accurate decisions and obtain control data earlier in the process (Kuhn and Johnson, 2013). Robustness of predictions is fundamental to predictive modeling. Computer programs assess the correlations between unrelated data, which may lead to erroneous predictions, *i.e.,* latent and indirect correlations from sensor data in a manufacturing process may lead to false conclusions of key variables influencing strength properties. Different model techniques are more applicable for different processes, *i.e.*, model techniques created to reduce the number of variables and reduce collinearity, *e.g.,* partial least squares (Özel and Karpat, 2005; Kuhn and Johnson, 2013). Model validation is fundamental to predictive analytics (Kuhn and Johnson, 2013).

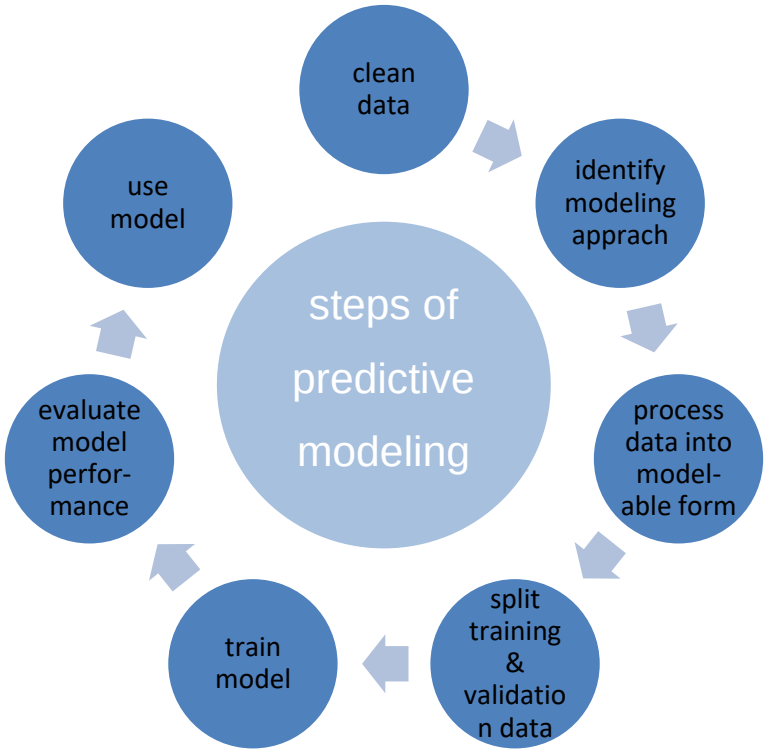

Figure 8: Seven steps of creating a predictive model (MathWorks Inc., 2019).

Key Methods for Predictive Modeling & Advanced Algorithms

Predictive modeling uses a variety of methods, and a description of some of the common methods used in this thesis are given below. A detailed review of each method is not presented given the breadth of literature on the subject. A general overview that highlights the unique contributions of each method is presented. The eight chosen predictive models (MLR, Rigde, LASSO, PCR, PLS, RT boosted, NN) provide a broad variety of usage, from linear relations to methods which address collinearity as well as methods which are able to handle multiple products.

Multiple Linear Regression (MLR)

Multiple linear regression (MLR) calculates linear correlations among the dependent variable and independent variables (Pearson, 1930) and predicts a dependent variable with an intrinsically linear function, Equation [1] is the primary goal (Kenton, 2019; SAS Institution Inc., 2019b). To get the best fit of the line, the least-squares method is typically used as a default. The least-squares method calculates a straight line through the data and minimizes the squared distances from the data points to the regression line. The input data assumes a linear relationship between the variables and the average of dependent variable assuming non-correlated independent variables with normally distributed residuals with a mean of zero and constant variance, *i.e.*, ~N(0, σ^2) (Flom, 2018; Kenton, 2019). The seven main assumptions for the MLR (Darlington and Hayes, 2017) are: 1) dependent variable is normally distributed; 2) the data has a linear relationship; 3) the variance of the residuals is constant; 4) the errors are independent (absence of autocorrelation); 5), all variables are random variables; 6) the explanatory variables should not be highly correlated (multicollinearity); 7) the residuals are normally distributed. Using incomplete data limits the robustness of MLR models even when the above assumptions are met (Flom, 2018). MLR is also sensitive to outliers or data that exert undue leverage on the fitted line (Flom, 2018).

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} \quad [1]$$

i = n observations

Y_i = dependent variable

X_i = explanatory variables

β_0 = y-intercept

β_p = slope coefficients for each x_i

ε = error term (residuals)

Ridge Regression

Ridge regression is useful (Seif, 2018) when the independent variables are highly correlated, or the number of observations is smaller than the number of independent variables (Saptashwa, 2018; Seif, 2018), which is known as 'poor dimensionality' of the dataset and is a common problem in the sustainable biomaterials industries. While multiple linear regression minimizes the sum of the squared error term, ridge regression minimizes the sum of square error term plus a penalty term (Equation [2]). The least-square method calculates the best fit of the coefficients and does not adhere to the stringent assumptions of the MLR. Ridge regression addresses the problem of multicollinearity (highly correlated predictors). Multicollinearity can increase standard errors of the regression coefficients, deflate t-tests, give false or nonsignificant p-values, or decrease the accuracy of predictions.

There are five sources of multicollinearity (Montgomery and Peck, 1982): 1) collection of a very small dataset in a narrow space, time, or area can create multicollinearity, which is not present in the original population; 2) physical constraints of the model or population (*e.g.,* physically, politically, or legally) can create multicollinearity; 3) over-defined models with more variables than

observations.; 4) using variables which are powers or interactions or other units of another variable; and 5) extreme outliers can cause or hide multicollinearity.

The simplest way to visualize collinearity is by pairwise scatterplots of the independent variables. A more accurate way to determine collinearity is the variance inflation factor (VIF). VIFs of ten or higher are indicative of collinear variables (Akinwande, Dikko, and Samson, 2015).

As previously stated, ridge regression minimizes the sum of the squared residuals plus the penalty (λ * *slope*^{2}), where $0 \leq \lambda \leq \infty$. If the λ =0, there is no penalty term, and ridge regression and MLR are equivalent. Lowering λ decreases the slope of the regression line. The slope only can get close to zero, but never zero. Ridge regression helps reduce variance by shrinking parameters and making the predictions less sensitive to the coefficients; this helps reduce the impact of multicollinearity (Hoerl and Kennard, 1970).

$$\sum_{i=1}^{M}(y_i\hat{y}_i) = \sum_{i=1}^{M}(y_i - \sum_{j=0}^{p} w_j * x_{ij})^2 + \lambda \sum_{j=0}^{p} w_j^2 \qquad [2]$$

M = instances

P = features

W = coefficients

I = observed values

Y = dependent variable

X = explanatory variables

λ = lambda, penalty term

LASSO Regression

Least Absolute Shrinkage and Selection Operator (LASSO) regression operates in a similar method as ridge regression but uses a different penalty calculation. In both cases, a biasing term (or penalty) reduces multicollinearity and overall model variance (Tibishirani, 1996). Ridge regression uses the squared coefficients, while LASSO regression uses absolute values bias (Equation[3]).The LASSO regression helps to reduce the over-fitting of the model and reduces irrelevant variables to zero (Saptashwa, 2018; Seif, 2018). Therefore, LASSO regression is a little bit better than ridge regression at reducing variance in models that contain a lot of irrelevant variables. In contrast to ridge regression, LASSO regression tends to perform better when most variables are useful. The exclusion of irrelevant variables via the LASSO regression makes the final equation simpler and more comfortable to interpret (Tibishirani, 1996).

$$\sum_{i=1}^{M}(y_i\hat{y}_i) = \sum_{i=1}^{M}(y_i - \sum_{j=0}^{p} w_j * x_{ij})^2 + \lambda \sum_{j=0}^{p}|w_j| \qquad [3]$$

M = instances

P = features

w = coefficients

I = observed values

Y = dependent variable

X = explanatory variables

λ = lambda, penalty term

Principal Component Regression (PCR)

Principal component regression uses the principal components (PC) of the explanatory variables for the regression model. The principal components are calculated with the principal component analysis. Therefore, the data are plotted in a high dimensional coordinate system where the number of dimensions is equal to the number of variables. The data gets centered (averaged in every direction, and the center of gravity is shifted to 0). From the centered data, a vector matrix (*n*n*) is calculated with the variance and covariance of all variables. A linear transformation of the original dataset transforms data points into principal components, and the eigenvectors and eigenvalues are calculated.

The eigenvectors describe the direction of the vector, and the eigenvalue describes the magnitude. All vectors are perpendicular to each other. The eigenvalues get sorted by size, and the highest eigenvalue is the PC1. Dividing the eigenvalues by *n*-1 and summing all those up, gives the total explained variation. The sum of the eigenvalues divided by *n*-1 divided by the total variation provides the percentage of variation this principal component explains in this system (Pearson, 1901; Shlens, 2005).

The primary use of this method is to overcome collinearly (Wold, Esbensen and Geladi, 1987; Liu *et al.*, 2003). The other idea is to reduce the dimension of the regression, but it only calculates new values (principal components) out of all explanatory variables without eliminating unimportant ones (Liu *et al.*, 2003; Li and Wang, 2014). Another issue with PCA is that its variance determines the importance of a variable. Therefore, variables with high variance are treated as important, while variables with low variance are treated as noise. Given that PCA relies on the variance-covariance matrix, the data must be standardized given the presence of scale variations which makes it difficult to directly interpret the model relative to the ease of interpretation of the coefficients in an MLR model. Data are assumed to be linear and orthogonal to each other, and when the assumption is violated, PCA gives biased or incorrect results (Shlens, 2005; Goyal, 2018).

Partial Least Squares Regression (PLS)

Partial least squares regression (PLS) reduces the explanatory variables to a smaller uncorrelated set of components and performs a least-squares regression on the subset instead of all variables. PLS is useful when explanatory variables are collinear or when there are more explanatory variables than observations or when ordinary least squares regression fails or calculates coefficients with high standard errors. PLS does not assume that the dependent variables are fixed, unlike MLR. Therefore, predictors can be measured with error, making PLS more robust against measurement inaccuracies (Lorber, Wangen, and Kowalski, 1987).

The benefit of PLS is that it can fit several dependent variables in one model (Addinsoft, 2018). PLS can be seen as to interconnected PCA analysis, PCA(*X*) and PCA(*Y*). PLS regression models the dependent variables in a multivariate way, and results can vary significantly from individually calculated dependent variables. Figure 9 illustrates how the different variables load the first and second factor of the PLS regression.

When using PCA to reduce features, the principle is to maximize the variance of the features itself. PCA calculates those features as high-dimensional points and does not take their classification label into account. PLS, on the other side, uses the annotated label to maximize inter-class variance. The primary use for PLS is classification, while PCA is better for a simple and linear dimension reduction.

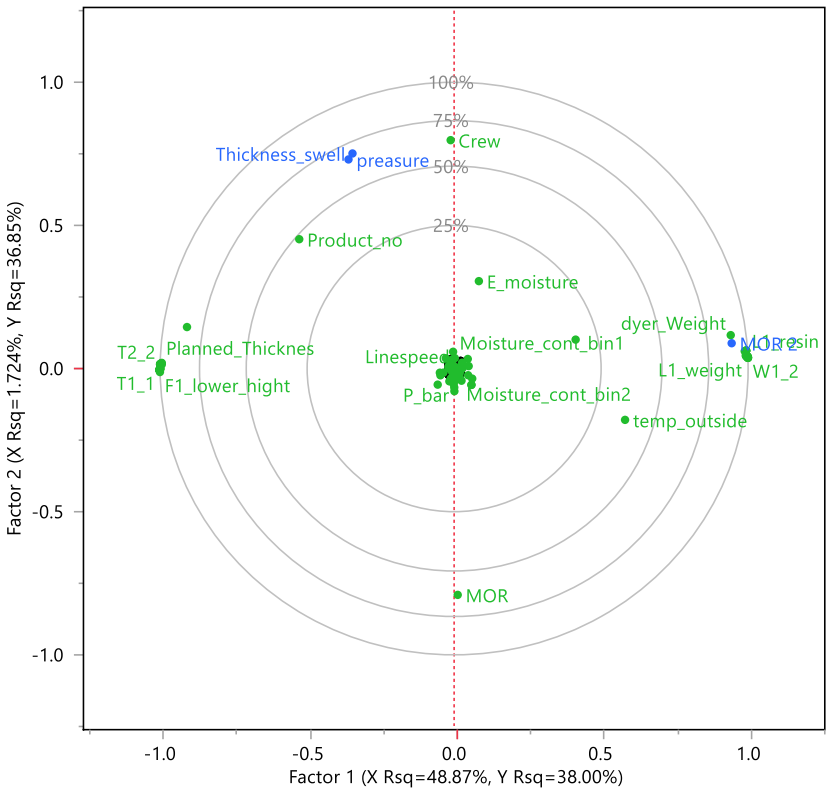

Figure 9: Partial least squares correlation loading plot of the simulated dataset for this study. Green dots show the loading and direction of the independent variables; the blue dots show the loading and direction of the dependent variables.

Boosted Regression Trees

Regression trees predict continuous dependent variables and are a subset of a much larger body known as decision trees. Regression develops nodes or leaves in the shape of decision trees. The split of each leaf or node is done to minimize the generalized variance associated with the sum of squares error (SSE) for each leaf. There are several techniques available to the analyst, and techniques are typically a function of the software capabilities. The splitting of a leaf to minimize the SSE can be based on the simple arithmetic average, simple linear regression, MLR, polynomial regression, quantile regression, etc.. All regression tree models have one or more input variables and one output variable (Figure 10).

Developing regression trees using bootstrapping (a resampling method) helps in prediction from regression trees (Efron, 1979). Boosted trees are still a form of 'greedy algorithms' and suffer from overfitting of the data which leads to poor validation (Sutton, 2005; Elith, Leathwick, and Hastie, 2008; Tufts, 2015).

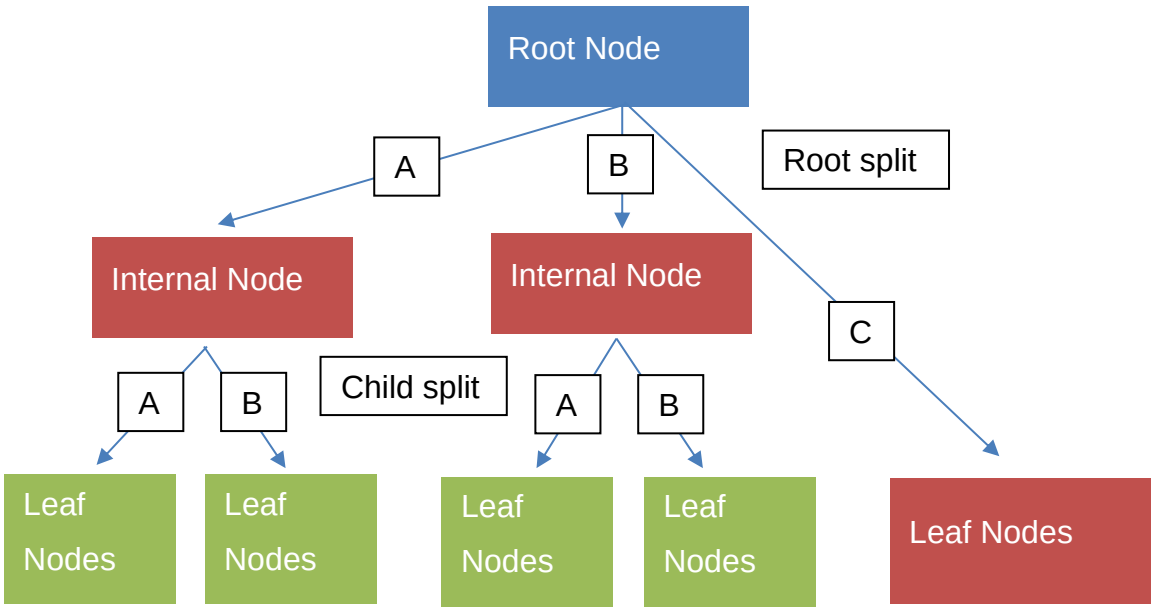

Figure 10: Simple decision tree model with one root split and two child splits (Quinlan, 1986).

Neural Networks (NN)

A neural network (NN) is a series of algorithms, modeled like the neurons of the human brain, designed to identify patterns (Bain, 1873). NN interprets information through machine observation, classification, or grouping data. These networks help to interpret images, classify text, compare documents, detect anomalies, or predict variables. An input, hidden, and output layer from a simple neural network. Nodes connect the layers and form a network of interconnected nodes (Figure 11) (Donges, 2018; skymind, 2018; SAS Institution Inc., 2019a).

To calculate the output, the input nodes must be entered into the system. All input nodes are multiplied by a factor (arrows in Figure 11) and added up in the hidden layer node. This procedure is done with every additional layer until it reaches the output layer. Input variables (nodes) have different impacts on the next layer. Therefore, the factor (arrows in Figure 11) can be from 0 (no impact) to infinity (extremely influential).

Compared to regression trees or other regression methods, NN are tough to interpret and are a type of ‘black box’ model, meaning there is a well-trained output, but there is no easy way to interpret the model.

With increasing computational power, NN can handle more and more variables and more massive datasets. The needed computational power for a NN is not only dependent on the data size, but also on the depth and complexity of the network created. For example, a NN with one layer and 50 neurons will be much faster than a bootstrapped forest with 1,000 trees. In comparison, a NN with 50 layers will be much slower than a bootstrapped forest with only ten trees.

One advantage of NN is the ability to model complex and non-linear relationships in the data. Unlike other prediction methods (MLR, PLS, Ridge, LASSO), NN do not have any restrictions on the input variables or residuals.

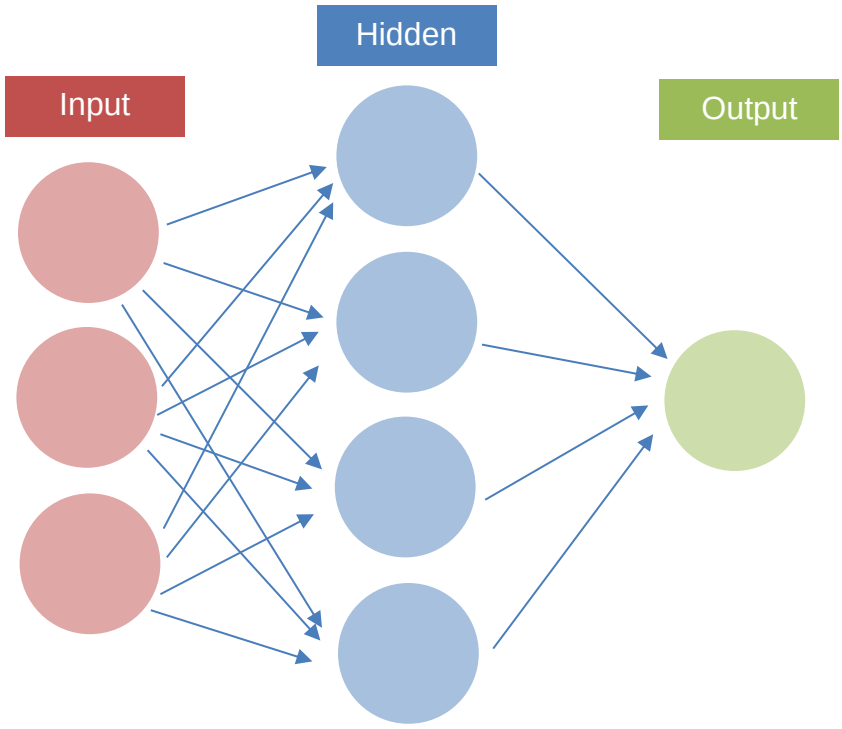

Figure 11: Simple neural network with input, hidden, and output layer connected via nodes (Bhadeshia, 1999).

Model Validation

Developing a robust model (extreme observations have little effect on the predictions and predictions are sustained with the same model) is the goal of any predictive modeling effort, and the validation is essential to prevent overfitting of the model and to find the best parameters (Kelley, 2017). Without validation, the model is perfectly fitted to the whole dataset, but the performance on new data is unknown (Raschka, 2018). The principle of validation is to train the model on just a part of the data and validate it on the other part of the data set. Through this split, the performance of new data can be measured (Kelley, 2017; Raschka, 2018).

The two most common validation methods are the hold-out and *k*-fold validation (Figure 12). The hold-out validation splits the entire dataset into two parts (common is 80% training and 20% validation or testing data sets) (Bronshtein, 2017). The model is trained or developed on the test data and validated on the validation data set. This method may still tend to overfit the data

because it is dependent on which data reside in the split datasets. A common second method is the *k*-fold method, where the data are split into *k*-folds (*k* = random chosen number). Each fold is one-time in the validation dataset, and all others are used as the training dataset. A benefit of *k*-folds is the model is trained and validated *k*-times, and therefore, assesses more variation in the dataspace and prevents overfitting. In contrary to the hold-out method, the *k*-fold method has to be calculated *k*-times, which is more computer-intensive (Zhao, 2016; ebc Inc., 2017; Kelley, 2017; Brownlee, 2018).

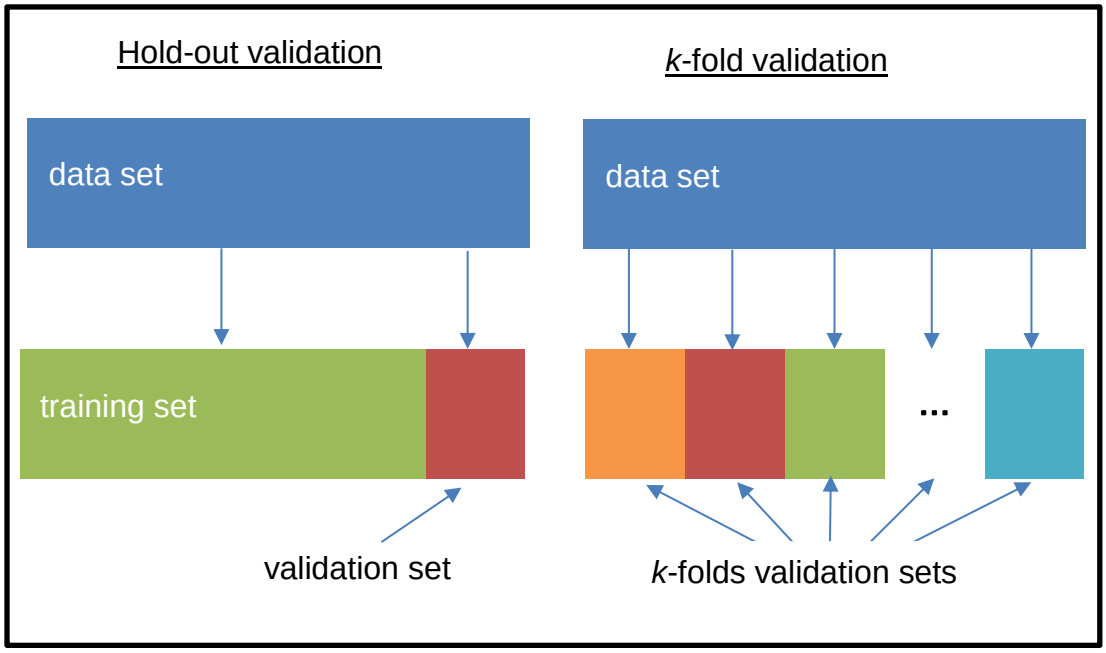

Figure 12: Illustration for hold-out and *k*-fold data set split and validation (Zheng, 2015).

CHAPTER THREE
MATERIALS, METHODS, AND TECHNIQUES

Concept of the Software Program

There are two main functions of the software program: 1) cleaning the database (fusing, aligning, quality assessing, imputing data); 2) and predicting destructive test data of manufacturing databases. The research will demonstrate the potential of a new data quality management method. The idea is an iterative program that processes every new input and predicts the associated quality control (QC) value (*e.g.,* destructive tests from a laboratory). There are two iterations of the program. The first iteration calculates all methods and models (to be explained in detail as follows), and the second iteration uses the calculated methods and models of the first iteration.

In the first iteration, every *X* seconds (preset value is five seconds, but can be individually changed), the program starts one of the two cycles. It recalculates if a new QC value is entered (“yes-loop” in Figure 13) and uses the calculations when no new QC value is entered (“no-loop” in Figure 13). In the ‘yes-case’ the program fuses and aligns all new data entries, looks at the data quality in the quality assessment part, selects and calculates an imputation method for every column in the database, imputes missing values, selects calculation variables, selects and calculates a predictive model for every QC variable, and predicts the new QC values.

If “no”, the program goes through the same steps without recalculating the imputation method and predictive model (Figure 13). In the following chapters, the detailed steps are described further.

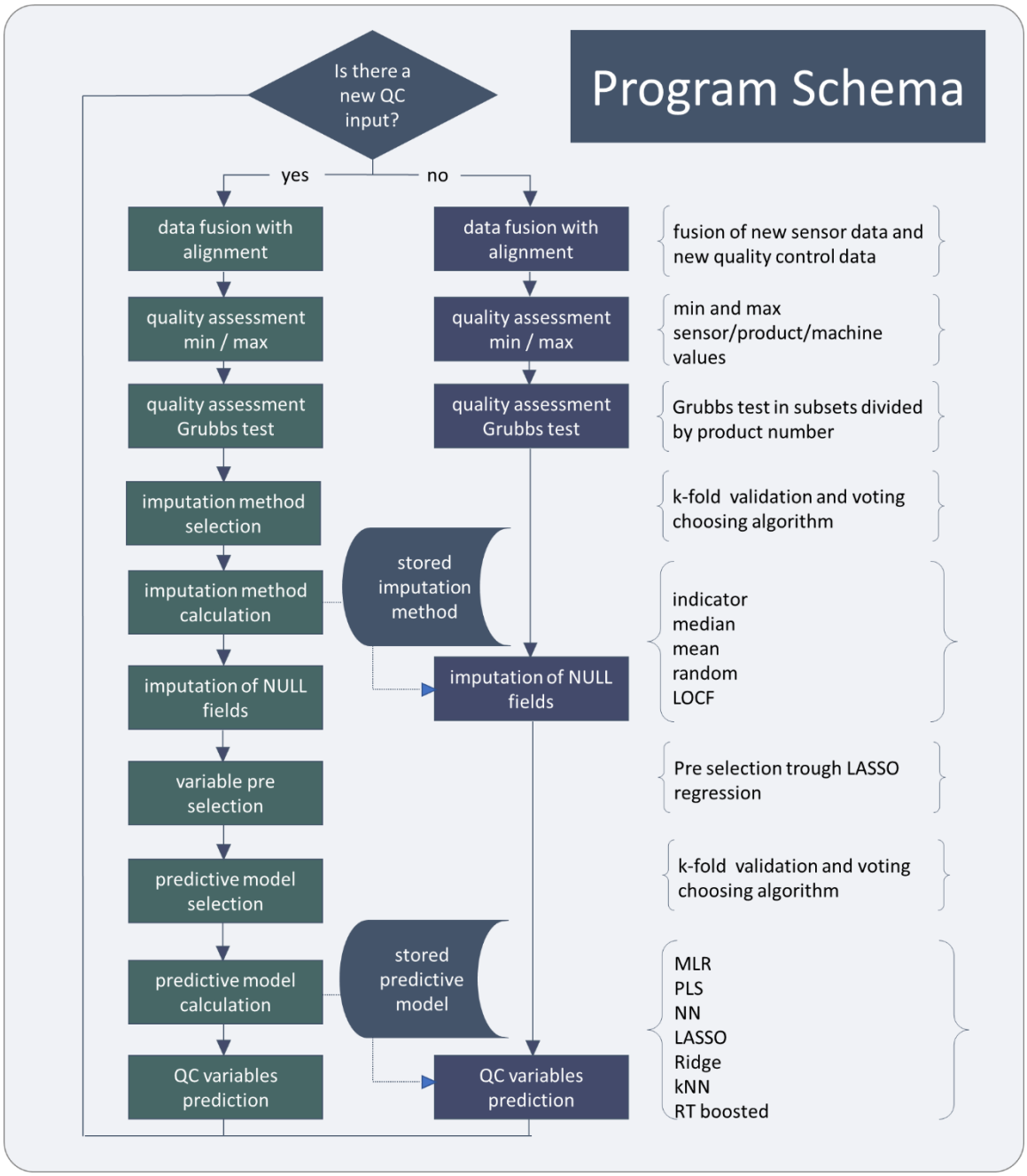

Figure 13: Concept of the program.

Computing Environment

The software system needs two software pieces to function, a MySQL® database for the data storage and Python® to run the code.

Database

The program uses the MySQL® 8.0 database from Oracle (www.mysql.com). The MySQL® database has the name “program” with the “root” as the username and the password “Root.” The database has five entities, two input entities, one clarification entity, and two output entities (see Figure 14). The “qualitycontrol” entity uses quality control measurements (*e.g.,* bending tests, thickness swell or all destructive tests) for the database. The timestamp is the primary key (the primary key is a unique value for each record of the table) for all input and output entities. The “qualitycontrol” entity needs an input number (inputNo) that automatically increments the next higher integer for every new input. The “inputNo” assures the program can detect a new input in the “qualitycontrol” entity. After those two variables, “TimeStamp” and “inputNo,” all quality control variables are in the “qualitycontrol” entity. Necessary for the program to function is the order of first the “TimeStamp”, second the “inputNo” and then all the quality control variables.

The second input table is the “sensordata” table with first the timestamp “STime” as primary key and the measured line speed as the second variable “Linespeed” followed by all sensor variables. The “sensordistance” entity holds the sensor name “SensorName” of every sensor in the system. Every sensor name has to be unique because “SensorName” is used as the primary key. The variable “Distance” holds the distance from the quality control measurement point to the sensor in the distance unit of the speed (*e.g.,* if the speed is measured in meter per second, the unit is meter).

Six variables declare minimum and maximum values of the sensors (the range the sensor can measure), of the produced product in the production line (range the product can be in, *i.e.*, the moisture content cannot be negative) and of

the machine (range the machine can operate). These minimum and maximum values are essential for the calculations of the quality assessment. One output entity is the “fuseddatameasured” entity. This entity has the same “TimeStamp” values as the “qualitycontrol,” which should be far less than the number of rows in the “sensordata” entity. The “TimeStamp” also is the primary key of this entity. The “inputNo” is also from the “qualitycontrol” entity, and the “Linespeed” is from the “sensordata” entity. “Assessed,” the fourth variable, is an integer between 0 and 3 that shows if the value was quality assessed. After the “Assessed” variable, the QC variables followed by the sensor variables are entered into the “fuseddatameasured” table. The “fueseddaraprediction” entity has the “TimeStamp” as a primary key, which is equal to the timestamps of the “sensordata” entity. The “Assessed” variable is the same concept as in the “fuseddatameasured” entity. The “Linespeed” values are the line speed values of “sensordata” entity at the same timestamp. After those three variables (“TimeStamp,” “Assessed” and “Linespeed”) the QC variables are in this entity, these values are not copied from the “qualitycontrol” entity, they get predicted in the process of the program. Sensor variables, which come from the “sensordata” entity, are last in the “fuseddatameasured” entity.

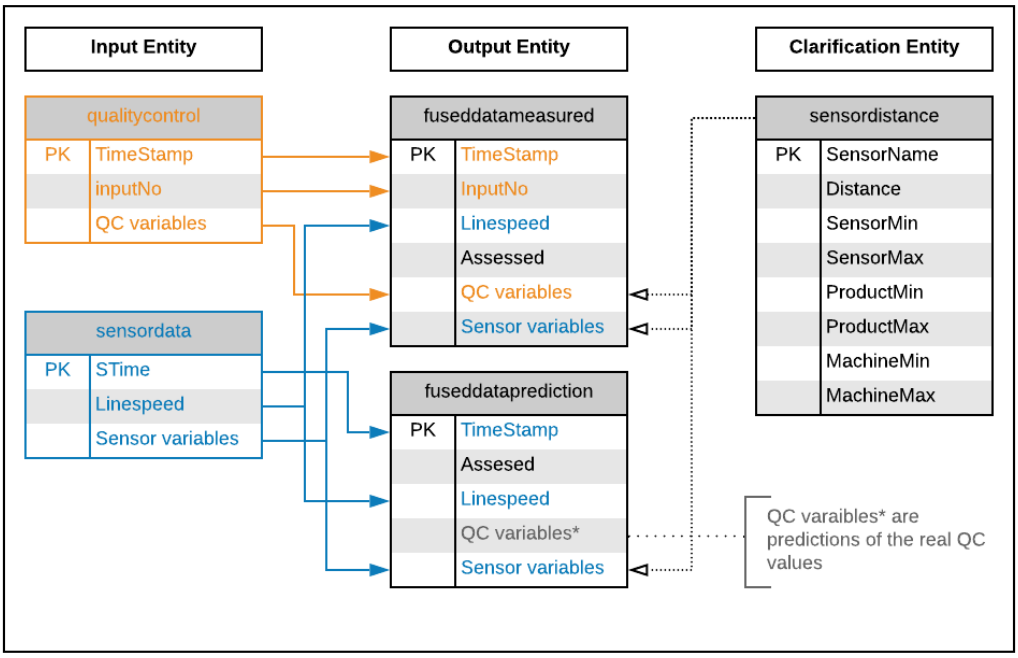

Figure 14: Schema of the database.

Python®

The code used for the program is written in Python® 3.7 (www.python.org). In order to run the program, the following extension modules of Python® 3.7 have to be installed: MySQL® connector, xlwt, xlrd, xlutils, sklearn, numpy, pandas, and outliers. The code is structured in 10 files (Table 3). The index file (Table 3: number 1) combines all other files' functions and includes the loop in which the program is running. Files two to ten (Table 3) include the functions for all the steps of the program. For the research, a separate report function and index file were created to ensure all the entities and calculation times are saved for every run in the research phase.

Table 3: File structure of the program.

No	File Name	Comment
1	Index.py / Index_research.py	depending on normal run or research run loads all functions and starts the loop
2	DataFusion.py	functions for data fusion
3	QualityAssessment.py	functions for Min/Max and Grubbs quality assessment
4	DataImputationMethod.py	choosing algorithm for data imputation
5	DataImputation.py	function for imputing data
6	VariableSelection.py	function for variable pre-selection
7	ModelMethod.py	choosing algorithm for predictive model
8	ModelMethodCalculation.py	function for training of predictive model
9	ModelImputation.py	function for predicting data
10	ResearchReport.py	needs a further document “research.xls”

Index File and Input Variables

The index.py file must be started to begin the program. This file loads all other files into the index file and starts the loop. Before running the program, the user can individualize variables at the beginning; these variables are described in Table 4.

In the first part of the index file, the variables get defined. Followed by the function definition “def Program()”, which defines and calculates the steps of the program according to Figure 13. The decision if the program should start the yes or “no-loop” is calculated by the comparison of the stored maximum value of the “inputNo” from the qualitycontrol entity to the maximum “inputNo” of this loop. If the “inputNo” are the same, the program starts the “no-loop,” if they differ the “yes-loop” (Figure 13) starts. The last part in the index file starts the “Program()” function into the endless loop.

Table 4: Changeable variables in the index.py file of the program.

Variable Name	**Preset value**	**Description**
secondsBetweenCheck	5	This value sets the interval after how many seconds the loop should restart
LinespeedMedianValues	10	This value looks at how many line-speed values are examined, and the median is placed into the two output entities
PercentTrainingsData	0.8	The percentage of how much data is in the training data set and validation data set for the imputation method and the predictive model validation
folds	10	A number of parts in which the dataset gets split and ensures every subset is included in the validation data set once at minimum.
LookBack	4000	The LookBack describes how many QC datapoints the programs should consider
Grubbsalpha	0.1	Alpha value for the Grubbs outlier test
NNAlpha	0.05	Alpha for the Neural network (NN)
HiddenLayers	(100,100,50)	The values in the "()" describe how many hidden layers the NN will have and how many neurons every layer has.
PCANoComponents	6	Value describes how many compounds the principal component analysis will calculate
RTbDepth	4	Describes the maximum splits of one regression tree
RtbEstimators	300	Describes how many weak trees the regressor will create
kNNNeigbors	3	The amount of nearest neighbors the *k* nearest neighbors regressor averages for the output
LASSOAlpha	0.5	Alpha value for the LASSO regression
RidgeAlpha	0.05	Alpha value for the Ridge regression
LassoCutOff	0	Absolut Lasso Coefficient cut of value for the variable pre-selection

Data Fusion with Alignment

Time-ordered data does not represent the measurements of one single product. Therefore, the data points have to be restructured via the dynamic time-lag (André *et al.*, 2008; Young *et al.*, 2014; Zeng *et al.*, 2016; Tian *et al.*, 2018). Simultaneously, the two input entities (sensordata and qualitycontrol) are fused to the two output entities (fueseddatameasured and fuseddatapredicted). The “fueseddatameasured” table takes all the new input times and the input number of the “qualitycontrol” entity (see Figure 15) and stores them in the “TimeStamp” and “inputNo” variables of the “fueseddatameasured” entity. The line speed gets added from the “sensordata” entity, where a median of *X* records (*X* is defined in the input variables as LinespeedMedianValues and preset to 10), with the next smaller or equal timestamps, represents the line speed for the “TimeStamp” in the “qualitycontrol” entity. Young *et al.* (2014) used the median of the last 100 inputs. The dependent variables from the “qualitycontrol” entity are added with the corresponding timestamp. The sensor values or independent variables from the “sensordata“ tables are calculated with a dynamic time-lag. For the dynamic time-lag, the distance of the sensor to the quality control measurement point is measured as well as the line speed for the specific record. With those two variables, the time difference between the two points can be evaluated (time = distance/speed). The variable “Assessed” is set to zero in this step of the program. The “fuseddatapredicted” entity gets filled with the new times of the “sensordata” entity and the median value of the line speed (same method as for the fueseddatameasured entity). The variable “Assessed” is set to “0” and the quality control variables to NULL. The sensor values for every record are calculated via the dynamic time lag (same method as for the “fueseddatameasured” entity).

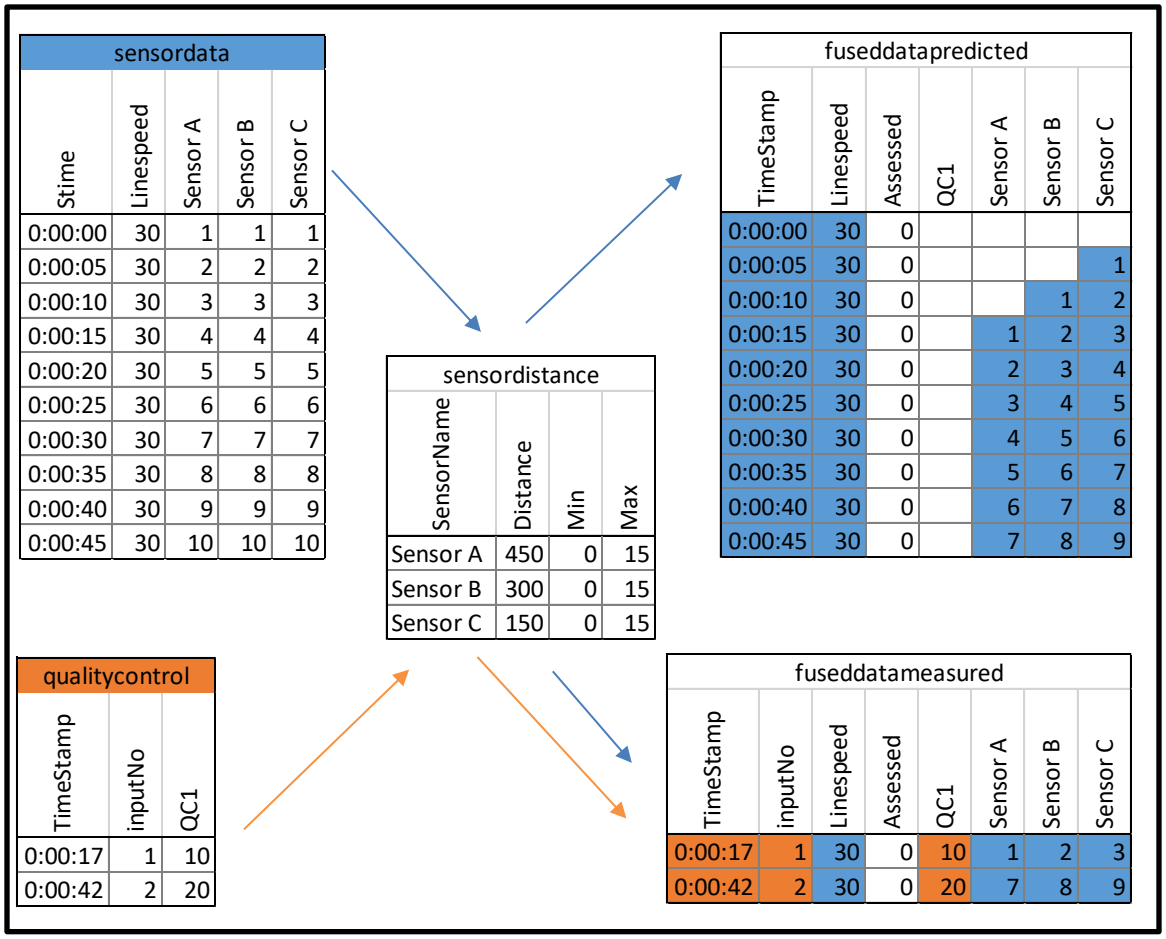

Figure 15: Data fusion and alignment example.

Quality Assessment

The quality assessment part of the program includes two calculations, first the minimum and maximum assessment and followed by an outlier test.

Minimum and Maximum Quality Assessment

Minimum and maximum quality assessments look at every value to see if it is under the minimum or over the maximum (Pipino *et al.*, 2002). If the value is in between those two values (minimum and maximum), the value stays in the database. If the value is outside the minimum and maximum values, the value gets deleted (Figure 16). The minimum and maximum values are stored in the "sensordistance" entity. Sensor, product, and machine minimum and maximum values are looked at and stored in the "sensordistance" entity. The values (minimum and maximum) against which the inputs are checked are the lowest of the three maximum values and the largest of the three minimum values. After every row is checked, the assessed value is updated to "1" in order to denote which rows were assessed for minimums and maximums.

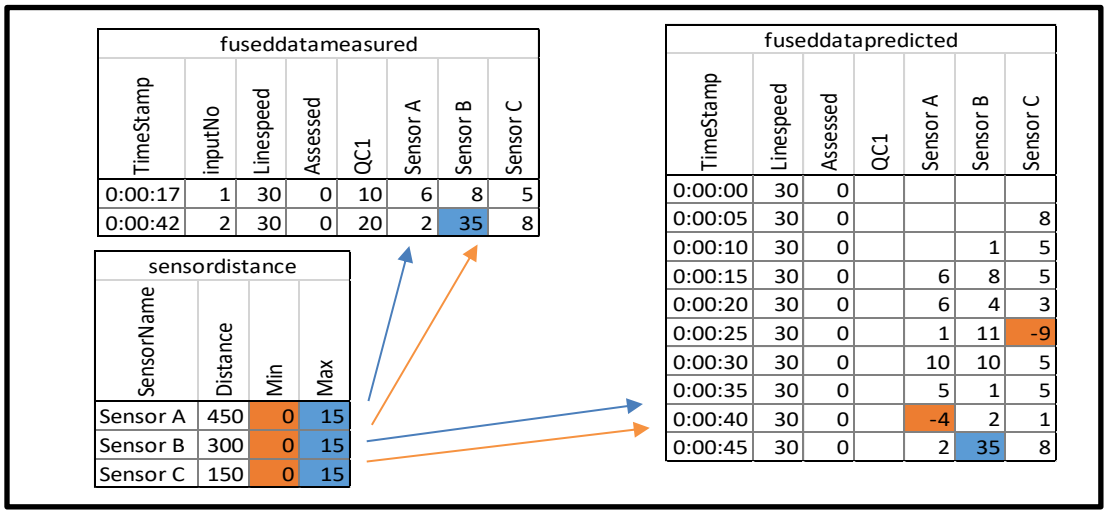

Figure 16: Schema of minimum and maximum quality assessment.

Grubbs Quality Assessment with and without Subsets

The Grubbs outlier quality assessment looks at every value and checks if the value is under the minimum, or over the maximum critical *X* value associated with the critical Grubbs value. There are two options to calculate the critical Grubbs value. The first option looks at the whole column at once; the second option looks at the column divided by the product number (variable "Product_no"). Both methods are programmed. To calculate the critical values, the dataset or subsets are put into the Equations $Grubbs_{crit}$ [4]and x_{crit} [5] (Grubbs, 1969). The critical *X* values are minimum and maximum values; all inputs in this range stay in the database, and all inputs outside the range are replaced with a NULL value. After every row is checked, the assessed value is updated to "2" in order to denote which rows underwent the Grubbs outlier assessment.

$$G_{crit} = \left| \frac{(n-1) * t_{crit}}{\sqrt{n * (n-2+t_{crit}^2)}} \right| \quad [4]$$

$$x_{crit(\min \max)} = \bar{x} \pm G_{crit} * s \quad [5]$$

G_{crit} = absolute critical Grubbs value

n = sample size

t_{crit} = probability of t-distribution with a certain α and degrees of freedom

x_{crit} = critical x values for a Grubbs test

$\bar{x}$ = mean of the sample

s = standard deviation of the sample

Data Imputation

There are several imputation methods for missing data points, mean imputation, median imputation, last observation carried forward (LOCF), random imputation, indicator imputation, expectation-maximization (EM), multiple imputation (MI), etc.. Zeng *et al.,* 2016 used various imputation methods (EM, MI, LOCF, Mean, Median) and improved the root mean square error of prediction (RMSEP Equation [6]). In this study, five imputation methods were programmed: indicator, median, mean, random, and LOCF imputation, due to the modular system, other methods can be easily added. The imputation methods are tested on the last *X* rows (*X* is defined in the input variables as “LookBack” and preset to 4000) of the “fueseddatameasured” entity. Every variable is tested for the most accurate imputation method, through the voting method (Chen *et al.*, 2009; Witten, Frank and Hall, 2011). Therefore, the dataset is split into *X* folds (*X* is defined in the input variables as folds and preset to 10) (Lachenbruch and Ray Mickey, 1968), the *X* folds split into a test dataset, and a validation dataset according to the preset percentage for those to datasets (the percentage is defined in the input variables as “PercentTrainingsData” and preset to 0.8).

There are *X* (fold) test datasets with associated validation datasets. Every method (indicator, median, mean, random, and LOCF) gets trained, see Table 5, on the training dataset and validated on the validation dataset. To validate the calculation, the RMSEP of every dataset, and the variable is calculated according to Formula [6]. Then the mean of the *X* (fold) RMSEP is calculated. Finally, the method with the smallest RMSEP is chosen as the imputation method. The imputation method for every variable is subsequently stored.

Every remaining NULL value in the “fuseddataprediction” (except QC variables, which get predicted) and “fuseddatameasurement” gets imputed with the previous calculated method. After every NULL value, the “Assessed” value for imputed rows is updated to “3”, this allows the program to distinguish which rows were imputed.

Table 5: Calculation of the five different imputation methods used in this study.

Method	**description**
indicator	Imputes an indicator instead of the NULL value, indicator is preset to 0
median	Imputes the median of the last x (LookBack) inputs
mean	Imputes the mean of the last x (LookBack) inputs
random	Imputes a random value in the range of this variable
LOCF	Imputes the last known observation into the missing field

$$RMSEP = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}} \qquad [6]$$

y_i = observed values

$\hat{y}_i$ = fitted values

n = number of observations

Variable Pre-selection

Predicting variables from a dataset with a lot of independent variables but a small amount of data records results in poor dimensionality of the dataset, and typically leads to predictive models that are not robust (André *et al.*, 2008; Zeng *et al.*, 2016; Tian *et al.*, 2018). Computational power generally increases with more independent variables. Zeng *et al.* (2016) used the LASSO coefficient ($\beta < 10^{-5}$) as the discriminatory variable to include or exclude the independent variable. André *et al.* (2008a) used a genetic algorithm to reduce the variable size. Tian *et al.* (2018) used stepwise regression and genetic algorithms for variable size reduction. Variable preselection increased the accuracy of predictions in previous studies (André *et al.*, 2008; Zeng *et al.*, 2016; Tian *et al.*, 2018).

In this study, the '*sklearn Python® module*' was used with the LASSO function. For this function, the last *X* (LookBack) rows of the entity "fuseddatameasured" are used. The rows with the independent sensor variables and dependent quality control variables are entered, and the LASSO regression is calculated on the full dataset. With the command "lasso.coef_," the β values of the regression (Equation [1]) are printed. If the absolute β values are lower than the cut-off *X* (*X* is defined in the input variables as "LassoCutOff" and preset to "0"), the variable will be excluded in the predictive modeling.

Predictive Modeling from the Fused and Data Quality Improved Database

Predictive models are created for every dependent variable as the last step of the program "yes-loop" (Figure 13). The approach used by Tian *et al.* (2018) which compared MLR, PLS, NN, decision trees, RT boosted, bootstrap forests, and Bayesian additive regression trees were followed. This approach built upon that by André *et al.* (2008b) which compared MLR and quantile regression (Young *et al.*, 2008), for the best performing method on a given dataset. Other studies mention comparisons of methods like PLS, orthogonal PLS, NN, and ridge

regression (André *et al.*, 2008; Zeng, 2011; André and Young, 2013; Tian *et al.*, 2018).

The predictive models are calculated and validated in the same way the data imputation method is calculated. First, the data are gathered from the "fuseddatameasured" entity and split into the validation folds. The training and validation datasets are then created. The quality control variables are the dependent variables and the sensor variables, are the independent variables. Multiple linear regression, partial least squares regression, neural network, principal component regression, regression trees boosted, *k* new neighbors , least absolute shrinkage and selection operator regression, and ridge regression are calculated in the program. The modules and functions used are listed in Table 6. For every model and dependent variable, the RMSEP are calculated and averaged over the folds. The smallest mean RMSEP will get the predictive model for the dependent variable. In the next step, the chosen model gets calculated on the whole dataset, and all NULL values of the dependent variables get predicted and imputed into the database.

Table 6: Python® modules of the predictive models.

Predictive Model	**Python® Module**
MLR	module: sklearn, function: LinearRegression
PLS	module: sklearn, function: PLSRegression
NN	module: sklearn, function: MLPRegressor
PCR	module: sklearn, function: PCA
RT boosted	module: sklearn, function: DecisionTreeRegressor, AdaBoostRegressor
*k*NN	module: sklearn, function: KNeighborsRegressor
LASSO	module: sklearn, function: Lasso
Ridge	module: sklearn, function: Ridge

Practical Benefits of the Program

The program helps to monitor the quality control variables without time delay, in an interval of seconds and as real time information. Quality control measurement are normally done 2-4 times per shift. Those QC measurements are done in a laboratory and take time. With the program an algorithm gets trained with all the QC measurements. Through this algorithm the time delay, between the taking a sample and getting the result, of more than two hours can be reduced to seconds. By monitoring the QC predictions on control charts, out of control points and trends can be visualized and addressed timely. The additional information between the measurements helps to figure out the true distribution of the data and highlight variables and QC variables where the targets are higher than needed. The program overall is a tool for calculating the QC variables, with further analysis and visualization of the data, potential optimizations can be determined.

Automated Report

For research and comprehensibility, an automated report of the last selection cycle is saved in global Python® variables. The report includes the imputation method validation with the RMSEP and NRMSEP (normalized RMSEP, Equation [7]) of every variable and fold, as well as the RMSEP and NRMSEP of the predictive modeling validation. The NRMSEP is calculated to eliminate the scale between different variables, due to the calculation method with the range, the NRMSEP is very sensitive to outliers. Outliers may bias the NRMSEP. The report automatically updates as soon as a new '*yes-cycle*' (Figure 13) is started. An example report for one variable of the simulated dataset can be seen in Table 7 and Table 8.

$$NRMSEP = \frac{RMSEP}{x_{max} - x_{min}} \quad [7]$$

Table 7: Example report of the *k*-fold cross-validation and voting method for one variable of the simulated dataset. For all five imputation methods, the RMSEP and NRMSEP of the validation datasets are printed. The lowest average is the imputation method for this variable.

Validation data of Variable: L1_hardener										
	Indicator		Median		Mean		Random		LOCF	
fold	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	1.51	22%	0.59	9%	0.61	9%	0.58	8%	0.26	4%
2	2	10%	0.82	4%	0.56	3%	0.2	1%	0.56	3%
3	2.21	10%	1.04	5%	0.83	4%	1.1	5%	0.9	4%
4	1.88	16%	0.66	6%	0.67	6%	0.68	6%	0.85	7%
5	1.91	10%	0.7	4%	0.7	4%	0.65	3%	0.25	1%
6	1.82	7%	0.81	3%	0.81	3%	0.53	2%	1.15	5%
7	1.01	6%	0.53	3%	0.68	4%	0.75	5%	0.33	2%
8	1.35	17%	0.23	3%	0.3	4%	0.67	8%	0.03	< 0%
9	1.33	9%	0.26	2%	0.33	2%	0.76	5%	0.26	2%
10	1.33	13%	0.26	3%	0.33	3%	1.2	12%	0.42	4%
Average	1.64		0.59		0.58		0.71		0.5	
Std dev	0.38		0.27		0.2		0.28		0.36	

Table 8: Example report of the *k*-fold cross-validation and voting method for one variable of the simulated dataset. For all eight predictive models, the RMSEP and NRMSEP of the validation datasets are printed. The lowest average is the predictive model for this variable.

Validation data of Variable: Thickness_swell_																
	MLR		PLS		NN		PCR		RT boosted		kNN		LASSO		Ridge	
fold	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	6.59	12%	3.68	7%	3.47	13%	5.62	10%	1.33	5%	4.02	15%	3.4	13%	3.4	13%
2	5.64	10%	3.66	6%	2.64	10%	5.73	10%	1.38	5%	4.23	16%	3.41	13%	3.41	13%
3	5.37	10%	3.49	6%	2.89	11%	5.59	10%	1.4	5%	4.08	16%	3.21	12%	3.21	12%
4	4.29	8%	3.41	6%	3.06	12%	5.58	10%	1.34	5%	4.24	16%	3.06	12%	3.06	12%
5	4.31	8%	3.49	6%	2.81	11%	5.66	10%	1.28	4%	4.11	16%	3.09	12%	3.09	12%
6	7.18	13%	3.65	6%	3.63	14%	5.86	10%	1.22	4%	4.22	16%	3.28	13%	3.28	13%
7	5.95	11%	3.48	6%	3.34	13%	5.66	10%	1.35	5%	4.29	16%	3.17	12%	3.16	12%
8	3.15	6%	3.47	6%	3.45	13%	5.38	10%	1.44	5%	4.14	16%	3.16	12%	3.15	12%
9	3.25	6%	3.49	6%	2.9	11%	5.57	10%	1.4	5%	4.17	16%	3.25	12%	3.25	12%
10	4.53	8%	3.51	6%	3.61	14%	5.5	10%	1.36	5%	4	15%	3.27	12%	3.26	12%
Average	5.03		3.53		3.18		5.62		1.35		4.15		3.23		3.23	
Std dev	1.35		0.1		0.36		0.13		0.06		0.1		0.12		0.12	

Dataset Simulation

Two hundred and one simulated datasets are used to study the effect of missing values, outliers, the number of independent variables, the ratio of destructive test records to sensor records, and the effect of variable pre-selection. For this reason, a master dataset was created with 100 dependent variables, three independent variables and 4,500 records, including six distributions (Gaussian, Chi², F, T, Lognormal, Gamma), five products, two crews, and random errors, but no outliers or missing values. The master dataset was created in Microsoft Excel® and with the formulas according to Appendix 2. An example of a simulated distribution is shown in Figure 17. This variable has a Gaussian distributed with a standard deviation of 1.03 multiplied with the average of the five records T2_1, T2_2, T2_3, T2_4, and T2_5. This calculation simulates a distribution and correlation to other variables.

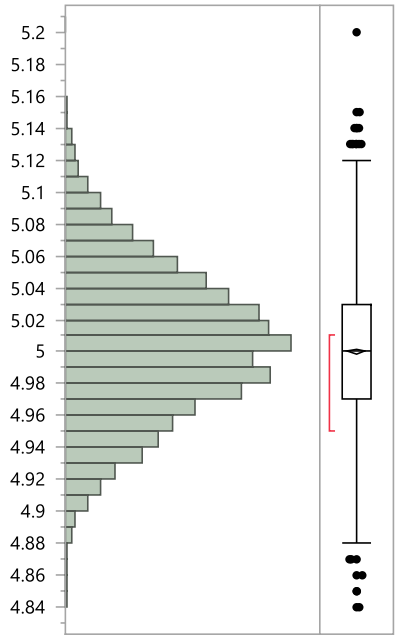

Figure 17: Example frequency density plot of the simulated variable dryer_airflow.

Through a Microsoft visual basic code (Appendix: VBA Code for Data Simulation), the VBA program imputes a defined percentage of random imputed NULL values, a defined percentage of NULL values in a block, and the same for outliers into the master dataset (exact values follow in the next chapter). Outliers were defined in the program as a random value between the minimum and maximum value of the variable (saved in the “sensordistance” entity). With these four factors, random NULL, blocked NULL, random outliers and blocked outliers, all needed datasets for the following tests for impact could be simulated.

Impact Tests of Different Factors

One of the research questions of this thesis focuses on the impact of outliers, missing data, sensor amount, and quantity of dependent records on the database and the predicted values and the impact that the program had on the predictive performance of the modeling function of the program. Therefore, four test sets where created, one to see the impact of missing values, one for the impact of outliers, one for the impact of the number of independent variables, and one for the impact of the number of dependent records. With these four tests, run rules for including or excluding variables should be created and used for the comparison on the industrial dataset.

Effect of Missing Values

Different percentages of missing values were added to the master dataset, via the VPA program, to see the impact on the RMSEP and the effectiveness of the program through the comparison of the untreated and the treated dataset. Two setups were created for this research. The first setup looks at the impact of missing 10% to 90% of values in 10% increments, with smaller increments between 0% and 10%. The exact percentages of included random distributed NULL values and blocked NULL values for all “missing data” tests are in Table 9. For the missing data tests, 445 quality control records were used. All other variables were set preset to the values in Table 9.

The datasets are run one time with the alignment, quality assessment, imputation, and variable pre-selection as well as one time without. The results of the predictions are compared with the simulated results from the master dataset and evaluated through the NRMSEP.

Table 9: Exact percentages of included random distributed NULL values, blocked NULL values, and outliers for the “missing data”- tests.

Dataset Name	**% Missing**	**% Random NULL**	**% Blocked NULL**	**% Random Outliers**	**% Blocked Outliers**
md_0	0	0	0	0	0
md_0.00105	0.105	0.084	0.021	0	0
md_0.003	0.3	0.24	0.06	0	0
md_0.007	0.7	0.56	0.14	0	0
md_0.014	1.4	1.12	0.28	0	0
md_0.023	2.3	1.84	0.46	0	0
md_0.04	4	3.2	0.8	0	0
md_0.05	5	4	1	0	0
md_0.06	6	4.8	1.2	0	0
md_0.08	8	6.4	1.6	0	0
md_0.1	10	8	2	0	0
md_0.2	20	16	4	0	0
md_0.3	30	24	6	0	0
md_0.4	40	32	8	0	0
md_0.5	50	40	10	0	0
md_0.6	60	48	12	0	0
md_0.7	70	56	14	0	0
md_0.8	80	64	16	0	0
md_0.9	90	72	18	0	0

Effect of Outliers

Similar to the missing values test the setup for the outlier tests are created. From 0% to 90% outliers replaced the real values, with the exact percentages of random and blocked outlier percentages in Table 10. An outlier was defined as a random value between the minimum and maximum value out of the “sensordata” entity of a variable. Otherwise, the minimum and maximum quality assessment would eliminate the value. For the outlier data tests, 445 records were used for the “qualitycontrol” entity. All other variables are set to the preset values in Table 9. The datasets are run one time with the alignment, quality assessment, imputation, and variable pre-selection as well as one time without. The results of the predictions are compared with the simulated results from the master dataset and evaluated through the NRMSEP.

Table 10: Exact percentages of included random distributed outliers, blocked outlier, and NULL values for the outlier tests.

Dataset Name	**% Outliers**	**% Random NULL**	**% Blocked NULL**	**% Random Outliers**	**% Blocked Outliers**
od_0	0	0	0	0	0
od_0.1	10	0	0	8	2
od_0.2	20	0	0	16	4
od_0.3	30	0	0	24	6
od_0.4	40	0	0	32	8
od_0.5	50	0	0	40	10
od_0.6	60	0	0	48	12
od_0.7	70	0	0	56	14
od_0.8	80	0	0	64	16
od_0.9	90	0	0	72	18

Effect of Quantity of Independent Variables

All sensors are ordered by the distance to quality control measurement point and added from the beginning by the furthest sensor in steps of +10 sensors a time. This looks at how early in the process, the program can produce accurate predictions. For every one of those datasets, 2% NULL values, and 2% outliers where added (Table 11). For the sensor amount tests, 445 records were used for the “qualitycontrol” entity. All other variables are set to the preset values in Table 9. The datasets are run one time with the alignment, quality assessment, imputation, and variable pre-selection as well as one time without. The results of the predictions are compared with the simulated results from the master dataset and evaluated through the NRMSEP.

Table 11: Exact percentages of included random distributed outliers, blocked outliers, random distributed NULL values, blocked NULL values, and number of sensors for the sensor amount tests.

Dataset Name	**Number of Sensors**	**% Random NULL**	**% Blocked NULL**	**% Random Outliers**	**% Blocked Outliers**
pd_0.1	10	1.6	0.4	1.6	0.4
pd_0.2	20				
pd_0.3	30				
pd_0.4	40				
pd_0.5	50				
pd_0.6	60				
pd_0.7	70				
pd_0.8	80				
pd_0.9	90				
pd_1.0	100				

Effect of Quantity of Dependent Records

The impact of the quantity of dependent records is measured with 16 different quantities of dependent records. The test looks at quantities from 10 records to 4002 records in +445 (10%) increments with closer steps between 10 records and 445 records (Table 12). For all of those datasets, 2% NULL values and 2% outliers where added, in the same percentages as for the previous sensor amount tests. All variables are set to the preset values in Table 9. The datasets are run one time with the alignment, quality assessment, imputation, variable pre-selection as well as one time without. The results of the predictions are compared with the simulated results from the master dataset and evaluated through the NRMSEP.

Table 12: Exact quantities of dependent records and independent records to test the effect of the quantity of dependent records.

Dataset Name	Dependent Records	Independent Records
pd_0.0022	10	4476
pd_0.005	23	4476
pd_0.01	45	4476
pd_0.025	112	4476
pd_0.05	223	4476
pd_0.075	334	4476
pd_0.1	445	4476
pd_0.2	890	4476
pd_0.3	1334	4476
pd_0.4	1779	4476
pd_0.5	2224	4476
pd_0.6	2668	4476
pd_0.7	3113	4476
pd_0.8	3557	4476
pd_0.9	4002	4476

Validation of a Real Dataset

The impact of the missing values, outliers, quantity of independent variables, and quantity of dependent records are tested according to Chapter three: “Impact Tests of Different Factors.” Three further tests were performed on a real dataset form a manufacturing company. The first one does not exclude any data. The second one is aligned with previous research, and with more than 20% missing data, the variable gets excluded (Zeng *et al.*, 2016). The third test doubles the 20% cut-off from Zeng *et al.*, (2016) to 40%.

Table 13: Exact quantities of dependent records and independent records to test the effect of the quantity of dependent records.

	No selection	**More than 20% missing data**	**More than 40% missing data**
Variables exclusion with missing data higher than	100%	20%	40%
Independent records	4435	4435	4435
Dependent records	1004	1004	1004
Independent variables remaining	248	232	225

CHAPTER FOUR
RESULTS AND DISCUSSION OF TESTS ON SIMULATED DATASETS

Missing data, outliers, sensor quantity, and quantity of dependent records had different impacts on the NRMSEP. For all of the following tests, a dataset with 4476 records and 100 predictors for five different products and two different crews was used. Missing values and outliers were added through the VBA program according to the percentages in Chapter 3. Missing data and outliers were randomly added and added as blocks. Random outliers or missing values are not connected to data in another variable and considered as MCAR. The imputation methods and predictive models are validated within the process of the program, and the 201 report files are available on request and in Appendix (we will add to www.spc4lean.com website). These files contain the validation of the imputation method and predictive model for every variable, as well as a copy of all five entities.

Data Quality Improvement

The program improves the data quality with the minimum and maximum values quality assessment and the Grubbs outlier test on datasets divided by the product number. The average improvement of 20 datasets from an NRMSEP from 11.2% from the untreated variable to 8.5% on the treated variable. The production dataset has, on average, 2.3% missing data in each variable. With the program, those missing values get imputed, and 95.12% more complete records could be achieved (Table 14).

Table 14: Average NRMSEP of the QC1 variable (simulated thickness swell) and the average count of complete records from 20 datasets (test xy) and two treatments.

Treatment	NRMSEP	Complete Records
untreated	11.2%	593
treated	8.5%	4476

Predictability Improvement

Assessing data quality and imputing data improves the accuracy of predictions. The average NRMSEP of 201 datasets in the treated and untreated versions, decreased from 63.95% to 11.53%. The standard deviation of the predictions could be reduced on average by 93.1% (Table 15). Therefore, the program reduces the NRMSEP and the standard deviation significantly.

The Tukey-Kramer HSD indicated that NRMSEP means ($\alpha = 0.05$) of treated and untreated datasets are different, and Levene's test that the variances are unequal. For both tests, Tukey-Kramer HSD and Levene's test, the better results were gained from the treated dataset. These differences are displayed in the boxplot, Figure 18, with 218 records, the treated data in blue, and the untreated data in red. The NRMSEP of this dataset decreased by 69.53% by using the treated instead of the untreated data. The standard deviation decreased from 0.204 untreated to 0.042 treated (Table 16).

The same dataset as for the boxplots in Figure 18 was used for a scatterplot of the actual values vs. the predicted values. The scatterplot was done twice, once with the treated data and once with the untreated data. The R^2 of the fitted line increased from 0.76% from the untreated dataset to 93.99% in the treated dataset. The RMSEP shrank by 75.40%. Moreover, the fitted line of the treated data indicates the predictions are closer to the actual values than compared to the untreated data. The difference in the ranges between the XY scatter plots highlights the information loss that may occur given that records greater than 10 for the untreated data are deleted due to missing cells within the record.

Table 15: Average NRMSEP of the QC1 variables (simulated thickness swell, test xy_0.1_c, and xy_0.1_r) and average standard deviation of 201 datasets and two treatments.

	NRMSEP	Std. Dev.
treated	11.53%	10.9%
untreated	63.95%	158.8%

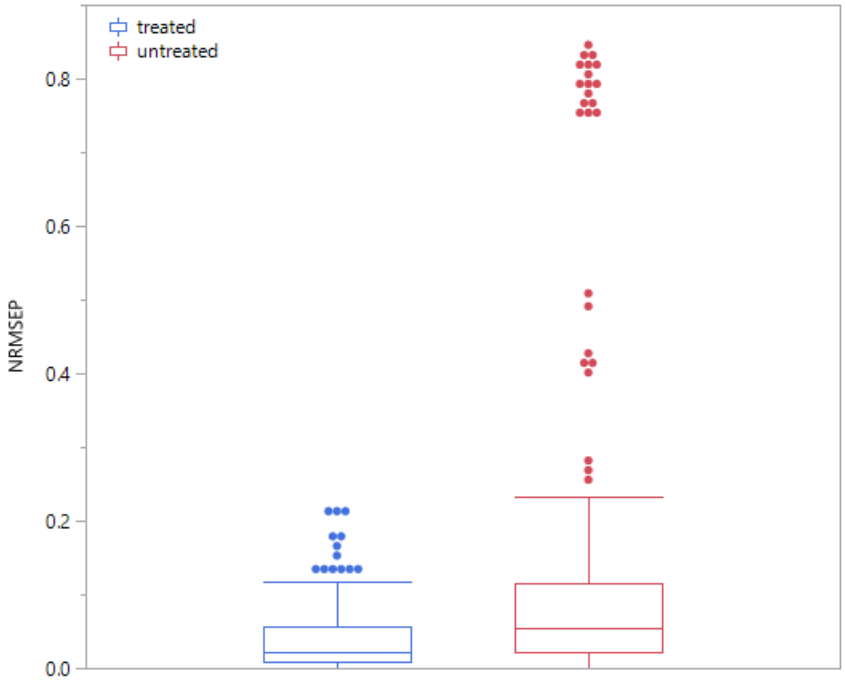

Figure 18: NRMSEP of the QC1 variable (simulated thickness swell, test xy_0.1_c and xy_0.1_r) divided by, treated with the program (red), and untreated (blue).

Table 16: Summary of statistics of the QC1 variable (simulated thickness swell test xy_0.1_c and xy_0.1_r) divided by, treated with the program, and untreated.

Summary Statistics	Treated	Untreated
NRMSEP	3.9	12.8
Std Dev	4.2	20.4
Std Err Mean	0.3	1.4
Upper 95% Mean	4.5	15.5
Lower 95% Mean	3.4	10.0
N	218	218

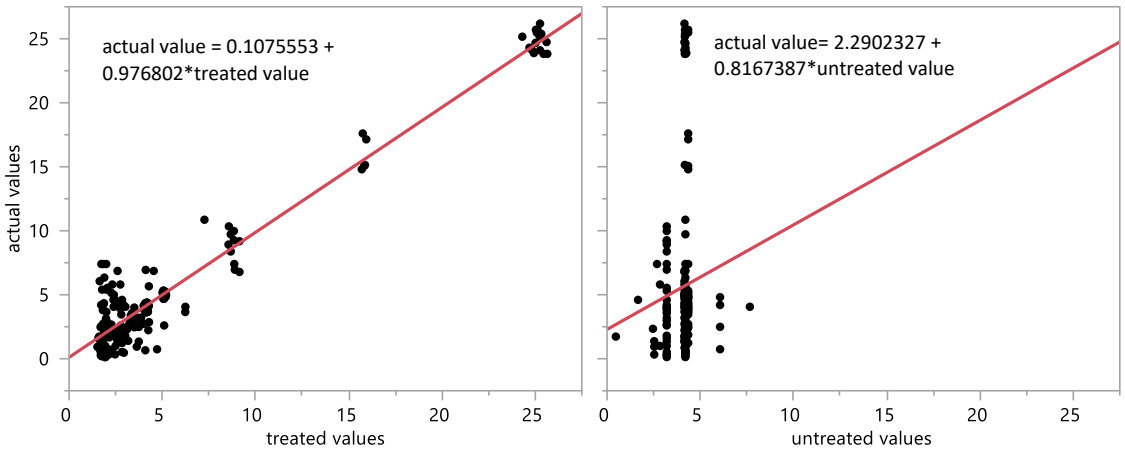

Figure 19: Scatterplot of real values vs. predicted values from the QC1 variable. Left is the treated data; right is the untreated data.

Table 17: Summary of fit of the scatterplot of real values vs. predicted values from the QC1 variable (simulated thickness swell test xy_0.1_c and xy_0.1_r) of the two treatments

Summary of Fit	Treated	Untreated
R^2	93.99%	0.76%
R^2 Adj	93.96%	0.30%
Root Mean Square Error	1.501	6.102
Mean of Response	5.596	5.596
N	218	218

Impact of Missing Data

Imputing missing data points increases the number of complete records. Sixteen datasets with 100 variables and different amounts of missing data were one time treated with the program and one time kept untreated. In the treated dataset, all records could be completed. Untreated datasets with more than 5% of missing data, had no complete records left. The probability of a complete record with 5% of random missing data is $0.95^{100} = 0.005$ or only 27 records out of the 4476 original records. Blocked missing values and random missing values decreased the number of complete records further, compared to the estimated number of the probability for MCAR (Table 18). The analysis of a real dataset shows that the average variable has 2.3% of outliers.

Table 18: Count of complete records of 16 datasets with 100 variables and 4476 records and different levels of MCAR data and blocked data, compared to the probability of MCAR.

% Missing Data	Complete Records		
	Treated	Untreated	Probability MCAR
0.0%	4476	4476	4476
0.1%	4476	3614	4050
0.3%	4476	2741	3314
0.7%	4476	1348	2217
1.4%	4476	465	1093
2.3%	4476	147	437
5.0%	4476	2	27
10.0%	4476	0	0
20.0%	4476	0	0
30.0%	4476	0	0
40.0%	4476	0	0
50.0%	4476	0	0
60.0%	4476	0	0
70.0%	4476	0	0
80.0%	4476	0	0
90.0%	4476	0	0

The average of ten datasets with 4476 records and 218 complete records, the NRMSEP of the treated data was 12.2% and of the untreated data 93.7%. The treated datasets had, on average, 95.12% more complete records than the untreated datasets. The NRMSEP of the overall 4476 records is 13.3%, when compared the 218 records of the untreated data to the same 218 records (Table 19 'same records') of the treated data, the NRMSEP rises by 1.1% and an increase of complete records, the standard deviation increases by 0.01 (Table 19). Therefore, through data imputation, the number of complete records increases by 95.12%, but also the NRMSEP increases by 1.1%, and the standard deviation increases by 0.01.

Each imputation method performs differently for one variable. Data imputation was tested on 100 datasets in the treated and untreated versions with 4476 records. Table 20 shows the validation data of one treated variable (Sensor18) with 4476 records. The average RMSEP differ throughout the five imputation methods (Indicator, median, mean, random, and LOCF). Table 21 shows the same variable untreated. Comparing the two tables, the RMSEP of the treated variable is lower for every method. The voting method with a *k*-fold cross validation chooses the method with the lowest average for this variable. In this case, the treated dataset imputes data with the LOCF imputation, and the untreated data imputes data with the median imputation. Therefore, quality assessed data performs better than untreated data.

Table 19: Average NRMSEP of the QC1 variable (simulated thickness swell, all md tests Table 9) and the standard deviation of 20 datasets with 4476 records and two treatments.

	NRMSEP		**Std. Dev.**	
	All Data	**Same Records**	**All Data**	**Same Records**
treated	13.3%	12.2%	17%	16%
untreated	93.7%	93.7%	250%	250%

Table 20: 10-fold imputation method validation of the variable Sensor18 (test: md_0.0023) of quality-assessed data.

Validation data of Variable:	Sensor18									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	1.51	2.0%	0.59	1.0%	0.61	1.0%	0.58	1.0%	0.26	<1.0%
2	2.00	1.0%	0.82	<1.0%	0.56	<1.0%	0.20	<1.0%	0.56	<1.0%
3	2.21	1.0%	1.04	1.0%	0.83	<1.0%	1.10	1.0%	0.90	<1.0%
4	1.88	2.0%	0.66	1.0%	0.67	1.0%	0.68	1.0%	0.85	1.0%
5	1.91	1.0%	0.70	<1.0%	0.70	<1.0%	0.65	<1.0%	0.25	<1.0%
6	1.82	1.0%	0.81	<1.0%	0.81	<1.0%	0.53	<1.0%	1.15	1.0%
7	1.01	1.0%	0.53	<1.0%	0.68	<1.0%	0.75	1.0%	0.33	<1.0%
8	1.35	2.0%	0.23	<1.0%	0.30	<1.0%	0.67	1.0%	0.03	<1.0%
9	1.33	1.0%	0.26	<1.0%	0.33	<1.0%	0.76	1.0%	0.26	<1.0%
10	1.33	1.0%	0.26	<1.0%	0.33	<1.0%	1.20	1.0%	0.42	<1.0%
Average	1.64		0.59		0.58		0.71		0.5	
Std dev	0.38		0.27		0.2		0.28		0.36	

Table 21: 10-fold imputation method validation of the variable Sensor18 (test: md_0.0023) of untreated data.

Validation data of Variable:	Sensor18									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	1.72	2%	0.73	1%	1.22	2%	53.32	74%	16.66	23%
2	5.07	2%	4.69	2%	4.63	2%	52.21	23%	12.79	6%
3	12.96	6%	12.69	6%	12.55	6%	54.75	24%	11.83	5%
4	12.06	15%	11.81	15%	11.7	15%	58.69	75%	12.84	16%
5	5.6	3%	5.27	2%	5.22	2%	53.37	24%	5.31	2%
6	5.59	3%	5.27	2%	5.22	2%	54.3	25%	7.01	3%
7	12.2	4%	11.95	4%	11.84	4%	53.62	19%	13.74	5%
8	13.24	5%	12.95	5%	12.8	4%	50	17%	14	5%
9	5.41	3%	5.03	3%	4.95	3%	51.31	31%	5.11	3%
10	1.64	2%	0.65	1%	1.29	2%	52.22	64%	0.92	1%
Average	7.55		7.1		7.14		53.38		10.02	
Std Dev	4.61		4.83		4.62		2.34		5.06	

Treating data, therefore, makes a difference and improves the NRMSEP. Table 22 lists the p-values of the Tukey-Kramer HSD test for mean comparisons of NRMSEP of 16 datasets with 4476 records. Tiny amounts of missing data (<1.4%) per variable do not change the NRMSEP significantly. Missing data reduces the number of complete records, with tiny amounts of missing data the reduction of the complete records seems to be very low and the predictability, in the form of the NRMSEP, seems to be unaffected. Although, the NRMSEP does not significantly change with very small amounts of missing data, the information loss with only 0.1% of missing data over 100 variables is 10%. In the presence of missing, data, imputation will prevent information loss, and with more than 1.4% missing data improve the NRMSEP. Treated datasets with more than 2.3% of missing data significantly decreased their NRMSEP compared to untreated datasets. This decrease seems to be caused by the information loss of over 90% of the untreated data. However, more than 5% of missing data on 100 variables did require data treatment because no complete records were left in the dataset. By imputing data, the records will be completed, and information loss is prevented.

Table 22: P-values from the Tukey-Kramer HSD test for mean comparisons of NRMSEP for treated versus untreated data of the missing data test. Compared are 16 treated datasets and 16 untreated datasets.

Untreated Data: Percent of Missing Data	Treated Data: Percent of Missing data										
	0	0.001	0.003	0.007	0.014	0.023	0.05	0.1	0.2	0.3	0.4
0	0.8166	0.9516	0.5612	0.062	0.1162	0.0077	<.0001	<.0001	<.0001	<.0001	<.0001
0.001	0.5487	0.7415	0.3567	0.0351	0.0666	0.0045	<.0001	<.0001	<.0001	<.0001	<.0001
0.003	0.5466	0.416	0.7252	0.5662	0.7181	0.2504	0.0026	<.0001	<.0001	<.0001	<.0001
0.007	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001
0.014	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001
0.023	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001
0.05	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
0.1	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
0.2	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
0.3	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
0.4	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
0.5	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
0.6	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
0.7	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
0.8	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
0.9	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

Impact of Outliers

Assessing data quality improves the accuracy of the variables. The average of five datasets with 100 variables and 4476 records, the NRMSEP of the variables to the actual values decreased from 11.2% for the untreated data to 8.5% for the treated data. The NRMSEP rises when the number of outliers increases. Not treating data is only advisable when no outliers are present in the dataset. Treating data impacts the data quality of the variables with outliers positively. The Grubbs outlier test was performed in two ways, once for the whole dataset at one and once with the data divided by the product number. The Grubbs outlier test on the whole dataset either improved the data slightly or had no significant effect. The outlier test on the data divided by the product number significantly improved the variables on average by 2% RMSE, except with zero percent outliers. In general, per 10% outliers, the RMSE of the variable increases by 0.05 (5%) (Table 23).

Each predictive model performs differently for each dependent variable. Predictive models were tested on 100 datasets in the treated and untreated versions with 4476 records. Table 24 shows the validation data of one treated dataset with 4476 records; the eight average NRMSEP differ throughout the predictive models (MLR, PLS, NN, PCR, RT boosted, kNN, LASSO, Ridge). Table 25 shows the same variable untreated. Comparing the two tables, the NRMSEP of the treated variable is lower for every model. The same effect can be seen in Figure 20, and the treated data has an R^2 of 95.14% and the untreated an R^2 of 1.57%. The voting method, with *k*-fold cross-validation, chooses the model with the lowest average for this dependent variable. In this case, the treated dataset predicts data with the boosted regression tree as well as the untreated dataset. Therefore, once the data is quality assessed, aligned, and imputed, the performance is better than untreated data.

Table 23: NRMSEP of the five variables (bin_level1, Planned_Thickness, planned_Density, T1, IB_Dens of the tests od_0 – od_40) with five different amounts of outliers. Every dataset had 4476 data points.

% Outlier	Treatment	Mean	NRMSEP				
			bin_level1	Planned_ Thickness	planned_ Density	T1_1	IB_Dens
0%	untreated	0.02%	0.00%	0.02%	0.04%	0.02%	0.03%
	treated	0.06%	0.20%	0.00%	0.00%	0.08%	0.01%
	treated with subsets	0.38%	0.23%	0.11%	0.10%	0.54%	0.92%
10%	untreated	5.74%	7.16%	3.93%	3.54%	6.84%	7.23%
	treated	3.20%	0.26%	3.88%	3.49%	6.84%	1.51%
	treated with subsets	2.62%	2.18%	2.68%	2.39%	3.23%	2.61%
20%	untreated	11.42%	14.31%	7.65%	7.67%	13.72%	13.73%
	treated	11.40%	14.47%	7.61%	7.58%	13.71%	13.64%
	treated with subsets	7.34%	8.17%	6.10%	5.99%	8.56%	7.87%
30%	untreated	17.25%	22.06%	12.08%	10.67%	20.89%	20.55%
	treated	17.27%	22.27%	12.06%	10.55%	20.98%	20.48%
	treated with subsets	13.44%	17.08%	10.59%	8.87%	16.11%	14.56%
40%	untreated	21.77%	27.52%	14.70%	13.39%	26.61%	26.62%
	treated	21.76%	27.66%	14.69%	13.26%	26.60%	26.57%
	treated with subsets	18.96%	23.99%	13.52%	11.78%	22.99%	22.54%

Table 24: Example report of the *k*-fold cross-validation and voting method for one variable of the simulated dataset. For all eight predictive models, the RMSEP and NRMSEP of the validation datasets of the pd_0.2_C test are printed. The lowest average is the predictive model for this variable.

Validation data of Variable: QC1																
fold	MLR		PLS		NN		PCR		RT boosted		kNN		LASSO		Ridge	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	5.10	9.0%	5.29	9.0%	4.75	17.0%	5.35	9.0%	2.57	10.0%	3.23	12.0%	5.17	19.0%	5.11	19.0%
2	5.12	11.0%	5.38	10.0%	4.84	17.0%	5.44	10.0%	2.42	9.0%	3.10	11.0%	5.19	19.0%	5.12	20.0%
3	5.23	9.0%	5.57	10.0%	4.99	19.0%	5.60	10.0%	2.47	9.0%	3.33	12.0%	5.34	20.0%	5.25	20.0%
4	5.21	10.0%	5.54	10.0%	4.89	17.0%	5.57	10.0%	2.50	10.0%	3.46	13.0%	5.30	20.0%	5.23	20.0%
5	5.05	9.0%	5.32	9.0%	4.51	16.0%	5.32	9.0%	2.53	10.0%	3.39	12.0%	5.12	19.0%	5.07	19.0%
6	4.94	10.0%	5.17	9.0%	4.62	16.0%	5.15	9.0%	2.48	9.0%	3.28	12.0%	4.99	18.0%	4.96	18.0%
7	4.94	9.0%	5.20	9.0%	4.78	17.0%	5.19	9.0%	2.53	10.0%	3.11	11.0%	5.00	19.0%	4.95	18.0%
8	4.81	9.0%	5.09	9.0%	4.53	16.0%	5.22	9.0%	2.59	10.0%	3.32	12.0%	4.93	18.0%	4.84	18.0%
9	4.94	9.0%	5.28	9.0%	4.78	17.0%	5.40	10.0%	2.41	9.0%	3.37	12.0%	5.04	19.0%	4.96	18.0%
10	5.12	11.0%	5.45	10.0%	4.97	17.0%	5.45	10.0%	2.39	9.0%	3.41	12.0%	5.20	19.0%	5.12	19.0%
Average	5.05		5.33		4.77		5.37		2.49		3.3		5.13		5.06	
Std dev	0.13		0.16		0.17		0.15		0.07		0.12		0.14		0.13	

Table 25: Example report of the *k*-fold cross-validation and voting method for one variable of the simulated dataset. For all eight predictive models, the RMSEP and NRMSEP of the validation datasets the pd_0.2_R test are printed. The lowest average is the predictive model for this variable.

Validation data of Variable: QC1																
fold	MLR		PLS		NN		PCR		RT boosted		kNN		LASSO		Ridge	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	5.90	10.0%	5.93	11.0%	4.83	18.0%	6.07	11.0%	3.30	12.0%	4.07	15.0%	5.92	23.0%	5.90	22.0%
2	5.89	10.0%	5.88	10.0%	4.87	18.0%	5.99	11.0%	3.10	11.0%	4.00	15.0%	5.89	22.0%	5.89	22.0%
3	5.92	11.0%	5.90	10.0%	4.81	18.0%	6.05	11.0%	3.43	13.0%	3.90	14.0%	5.91	23.0%	5.92	23.0%
4	5.98	11.0%	5.99	11.0%	4.53	17.0%	6.07	11.0%	3.57	13.0%	4.00	15.0%	5.97	22.0%	5.98	23.0%
5	5.90	10.0%	6.01	11.0%	4.45	16.0%	6.43	11.0%	3.20	12.0%	3.93	14.0%	5.91	22.0%	5.90	22.0%
6	5.84	10.0%	5.93	11.0%	4.79	18.0%	6.49	12.0%	3.04	11.0%	3.88	14.0%	5.86	22.0%	5.84	22.0%
7	5.75	10.0%	5.75	10.0%	4.46	16.0%	6.00	11.0%	3.23	12.0%	3.95	15.0%	5.76	21.0%	5.75	21.0%
8	5.75	10.0%	5.75	10.0%	4.33	16.0%	5.98	11.0%	3.10	11.0%	4.08	15.0%	5.79	22.0%	5.76	21.0%
9	6.14	11.0%	6.14	11.0%	4.66	17.0%	6.32	11.0%	3.33	12.0%	3.90	14.0%	6.16	23.0%	6.17	24.0%
10	6.09	11.0%	6.10	11.0%	4.83	18.0%	6.24	11.0%	3.36	12.0%	4.15	15.0%	6.09	23.0%	6.09	23.0%
Average	5.92		5.94		4.66		6.17		3.26		3.99		5.93		5.92	
Std dev	0.13		0.13		0.2		0.19		0.16		0.09		0.12		0.13	

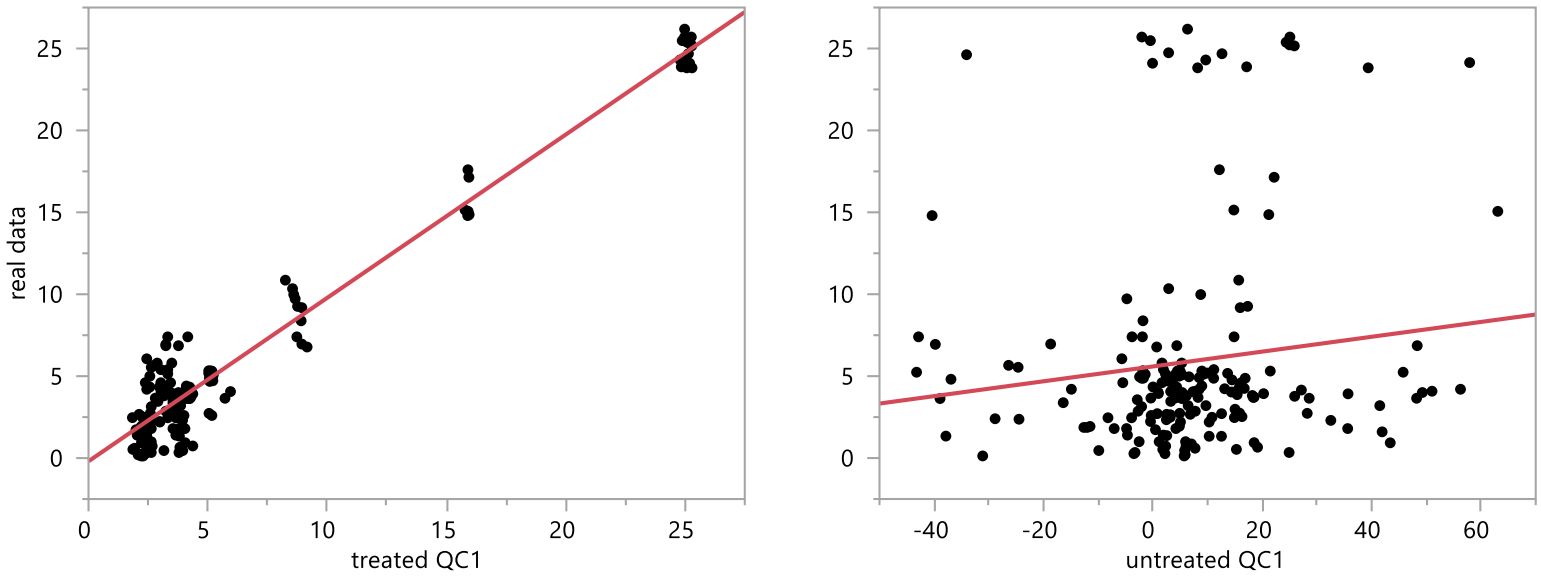

Figure 20: XY scatter plot of the validation datasets the pd_0.2_R in the treated and untreated version vs. the real data

Impact of the Quantity of Dependent Records

Treated data needs fewer dependent records (quality control data) to reach the same level of NRMSEP as untreated data. To achieve the same NRMSEP, the treated datasets needed 447 dependent records, while the untreated dataset needed 3133 dependent records. The NRMSEP was statistically equal for the treated dataset with 447 or more dependent records, according to Tukey-Kramer HSD. The number of dependent records was tested on 15 datasets with 4476 independent records and different quantities of dependent records. All 15 datasets were once treated and once untreated. With less data, the model seems to be too simplified, and the accuracy varies from test to test. The untreated data seems to have a lower NRMSEP with over 3133 dependent records, but the NRMSEP of the treated data includes all of the predictions. Table 19 showed that with the increase of 95.12% of complete records, the NRMSP increases by 0.11 (1.1%). The untreated data, with less than 3313 dependent input variables, have less than 130 complete dependent records, this seems to simplify the model and increase the NRMSEP. Therefore, the treated dataset needs seven times fewer dependent records than the untreated dataset to get accurate predictions, but those accurate predictions suffer from information loss of 95.12%.

Table 26: Count of dependent, independent and predicted records of 15 datasets and their NRMSEP of the treated and untreated dataset

Dependent Records	Independent Records	Predicted Records		NRMSEP	
		Treated	Untreated	Treated	Untreated
10	4476	4476	218	3.90%	24.89%
22	4476	4476	218	5.07%	12.63%
44	4476	4476	218	45.04%	13.43%
111	4476	4476	218	43.43%	28.24%
223	4476	4476	218	59.10%	30.42%
335	4476	4476	218	41.22%	52.79%
447	4476	4476	218	3.92%	59.75%
895	4476	4476	218	3.77%	68.58%
1342	4476	4476	218	4.06%	57.04%
1790	4476	4476	218	4.22%	43.49%
2238	4476	4476	218	4.32%	34.36%
2685	4476	4476	218	4.32%	27.78%
3133	4476	4476	218	4.49%	3.70%
3580	4476	4476	218	4.76%	3.71%
4028	4476	4476	218	4.65%	3.54%

Impact of the Quantity of Independent Variables

With every variable added the missing values accumulate, and the number of complete records is reduced. This relates to the probability of 0.98^x where x is the sensor quantity; the more sensors are added, the smaller is the probability for a complete record. The test was performed on one dataset with 4476 records, 2% of missing data, 2% of outliers, and the quantity of independent variables changed from 10 to 100 in +10 increments (Table 27). Adding sensors to an untreated dataset decreases the number of complete records and also the number of depended records the predictive model can be trained on. Therefore, the untreated dataset seems to perform reasonably between 30 to 60 variables, but with loss of records and information. When there are more than 60 variables applied to the model, the model of the untreated data seems to be too simplified through less complete records. Subsequently, the accuracy is not given. The treated dataset

has a five times higher NRMSEP with 10 and 20 variables, and from 30 to 100 variables, the NRMSEP is statistically equal according to the Tukey-Kramer HSD test. The first 20 variables are hardly correlated to the dependent variable; the first highly correlated variable is “Sensor24” with a correlation of 86%. This might indicate that as soon as a highly correlated variable is among the independent variables, the predictions improve.

Table 27: Count of complete records of 10 datasets with 4476 records and different quantities of independent variables (all pd tests table 12). The datasets have 2% missing value, and 2% outliers added.

Independent Variables	Complete Records		NRMSEP	
	Treated	Untreated	Treated	Untreated
10	4476	3317	19.83%	20.59%
20	4476	2391	18.49%	20.02%
30	4476	1752	4.56%	3.62%
40	4476	1291	4.50%	4.55%
50	4476	967	4.32%	5.64%
60	4476	778	4.31%	5.48%
70	4476	556	4.28%	53.60%
80	4476	430	4.33%	47.15%
90	4476	149	4.27%	45.48%
100	4476	112	4.97%	42.56%

CHAPTER FIVE
RESULTS AND DISCUSSION OF TESTS ON A REAL DATASET

Validation on a production dataset

The software program improved the number of complete records and accuracy of the predictions for both the simulated and actual production datasets. The actual production dataset already contained destructive test data fused with the sensor data that had multiple hours between the samples. Given that no real-time sensor data was provided between destructive lab samples, autocorrelation was not present in the actual production dataset. No dynamic alignment was done. The dataset was split into a sensor measurement entity and a quality control entity, where all sensor measurements were added to the sensor entity; 1004 of the measurements were added to the quality control entity. The 1004 represented approximately 100 records per product (Table 26).

Three setups are compared with no excluded variables, variables with more than 40% of missing data are excluded, and variables with more than 20% of missing data are excluded (refer to previously cited Table 13, page 61). All three setups are calculated in both the treated and untreated version. The number of complete records shrank by 90.5% for a dataset with 225 variables and by 90.9% for the dataset with 232 variables. In the treated version, all records were predicted. The untreated version could only predict records if variables with high missing rates (>60% of missing data) were removed (Table 28).

Table 28: Count of records and predictors of the three different tests of the production dataset divided by treatment.

	Complete Independent Records		Complete Dependent Records	
	Treated	**Untreated**	**Treated**	**Untreated**
248	4435	NA	1004	NA
232	4435	402	1004	91
225	4435	420	1004	96

The highest NRMSEP of the treated dataset was for all 248 variables. The lowest NRMSEP for the treated dataset had 232 variables (40% cut-off for missing values). The standard deviation for the NMRSEP of the treated dataset with 225 variables was 11.8% and was the highest of the three tests conducted (Table 28). The lowest standard deviation for the NMRSEP was for the treated dataset and was achieved with 232 variables (9.3%). The NRMSEP and the standard deviation for the NMRSEP for the untreated datasets were in all three tests data sets higher than the treated datasets. The scatterplots (Figure 21, Figure 22, and Figure 23) highlight the importance of treating data (Table 30). The test with 232 variables when treated improved the R^2 for predicted versus actual quality control characteristic from 33.21% to 76.37%, and the test with 248 variables improved the R^2 for predicted versus actual quality control characteristic from 1.92% to 75.74%.

The predictability improved for the simulated data when the number of complete records increased (Table 29). The untreated test with 225 variables had values which were far higher than all the other untreated or treated tests. The untreated 225 variables test chose the MLR as a predictive model; the Ridge regression performed better in all other cases. This may result in higher errors in the predictions.

Table 29: NRMSEP, standard deviation, and chosen predictive models for the three dependent variables of the real dataset with different numbers of independent variables.

Dependent Variable	Number of Variables	NRMSEP		Std. Dev.		Predictive Model	
		Treated	Untreated	Treated	Untreated	Treated	Untreated
QC1	248	7.12%	NA	9.9%	NA	Ridge	Ridge
	232	6.90%	9.91%	9.3%	13.0%	Ridge	Ridge
	225	7.11%	40.09%	11.8%	37.5%	Ridge	MLR
QC2	248	1.00%	NA	3.1%	NA	Ridge	Ridge
	232	0.92%	1.03%	2.8%	2%	Ridge	Ridge
	225	0.95%	91.5%	3.3%	23.3%	Ridge	MLR
QC3	248	5.10%	NA	7.0%	NA	Ridge	Ridge
	232	4.91%	10.27%	6.8%	22.4%	Ridge	Ridge
	225	4.93%	36.07%	6.9%	19.8%	Ridge	MLR

Table 30: Summary of Fit of the Scatterplots in Figure 21-23 with the data of the validation datasets of the QC1 variable with 3 different numbers of variables.

	248		232		225	
	treated	untreated	treated	untreated	treated	untreated
R^2	75.24%	NA	76.37%	33.21%	75.74%	1.92%
R^2 Adj	75.18%	NA	76.31%	33.04%	75.68%	1.67%
RMSEP	161.00	NA	157.30	264.43	159.37	320.43
Mean of Response	1802.80	NA	1802.80	1802.80	1802.80	1802.80
n	402	NA	402	402	402	402

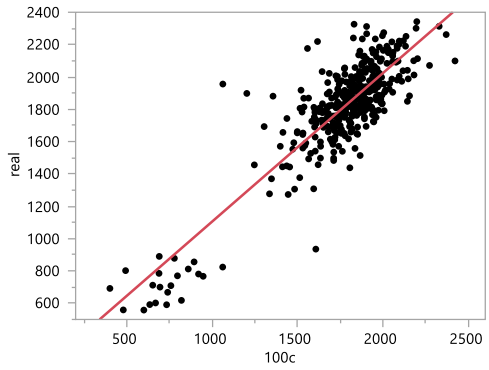

Figure 21: XY scatter plot of the validation datasets of the QC1 variable with 248 variables in the treated and untreated version vs. the real data (untreated were no complete records left)

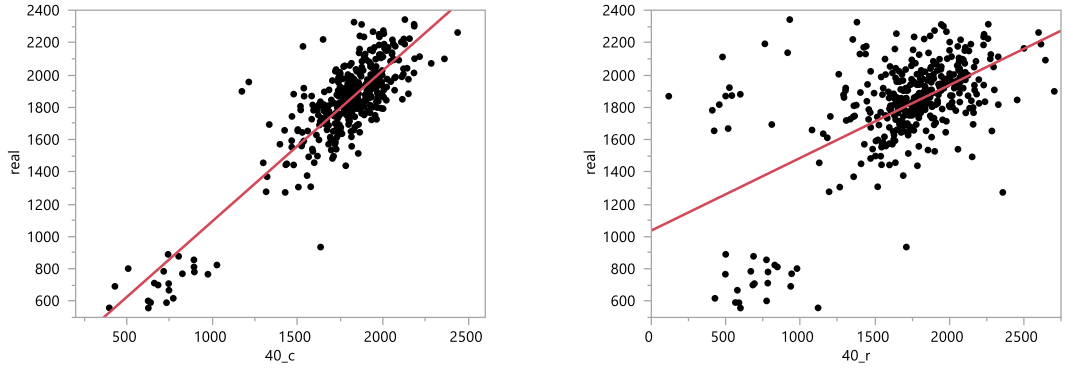

Figure 22: XY scatter plot of the validation datasets of the QC1 variable with 232 variables in the treated and untreated version vs. the real data

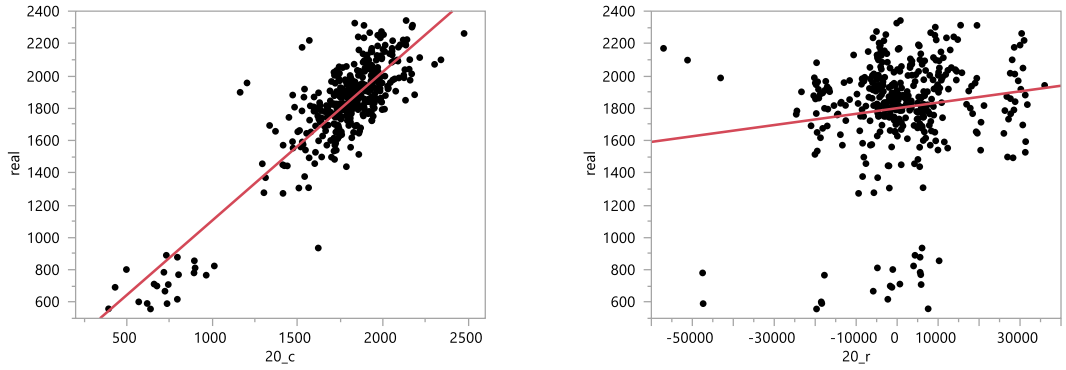

Figure 23: XY scatter plot of the validation datasets of the QC1 variable with 225 variables in the treated and untreated version vs. the real data

Validation using JMP

The software program in this study used Python® for the modeling and the code was validated with the commercial software JMP® Pro 14 from SAS (www.jmp.com). To validate the code, the same data calculated with the Python® program was used in JMP®. The datasets of the different methods were chosen according to Table 31. Of the 201 tests, PLS as well as the LASSO model were not chosen by the Python® program and therefore no reference data of the Python® model was existent to compare to the JMP® data. The calculation for an adaboost regression tree could not be found in JMP®.

MLR, PLS, PCR, *k*NN, LASSO, and Ridge regression were calculated in both programs. The predictions of the runs in Python® and the runs in JMP® of the same datasets were identical (Figure 24, Figure 25, and Figure 26). See the Python® code in Chapter 8 Appendix. Boosted regression trees were calculated in Python® and in JMP®, but the results differ slightly. For this test, the XY Scatterplot of the JMP® predictions and the Python® predictions had an R^2 of 99.87% (Figure 27). This difference is caused by the use of two different boosting methods, JMP® uses gradient boosting, while Python® uses adaptive boosting. The NN, in its nature, produces slightly different results with every recalculation of the data. Therefore, it is hard to compare the NN predictions of JMP® and Python® when the input the same dataset. For this test, the XY Scatterplot of the JMP® predictions and the Python® predictions had an R^2 of 98.45% (Figure 27).

Table 31: Validation of the predictive modeling python® code with the commercial software JMP® Pro 14

Predictive Model	Dataset Tested	Identical predictions	Comment
MLR	xy_0.01_c	yes	
PLS	Master dataset	yes	Master dataset without outliers or missing data, in the python program the only model to choose was set to PLS
PCR	xy_0.022_r	yes	
kNN	pd_0.1_c	yes	
LASSO	Master dataset	yes	Master dataset without outliers or missing data, in the python program the only model to choose was set to LASSO
Ridge	r_40_c	yes	
RT boosted	md_0.3_c	no	JMP only uses gradient boosting, the JMP® program used adaptive boosting (adaboost)
NN	od_0.07_c	no	

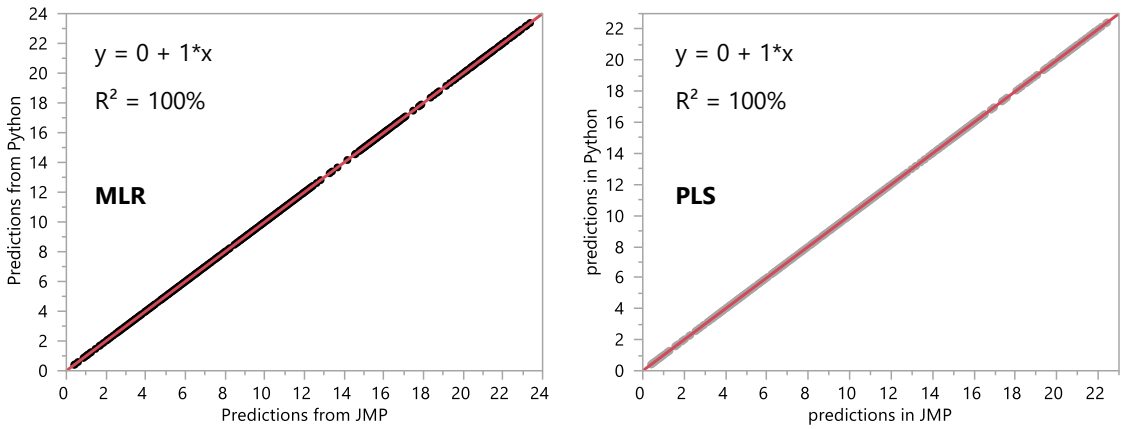

Figure 24: Comparison of the predictions of MLR and PLS from JMP and Python.

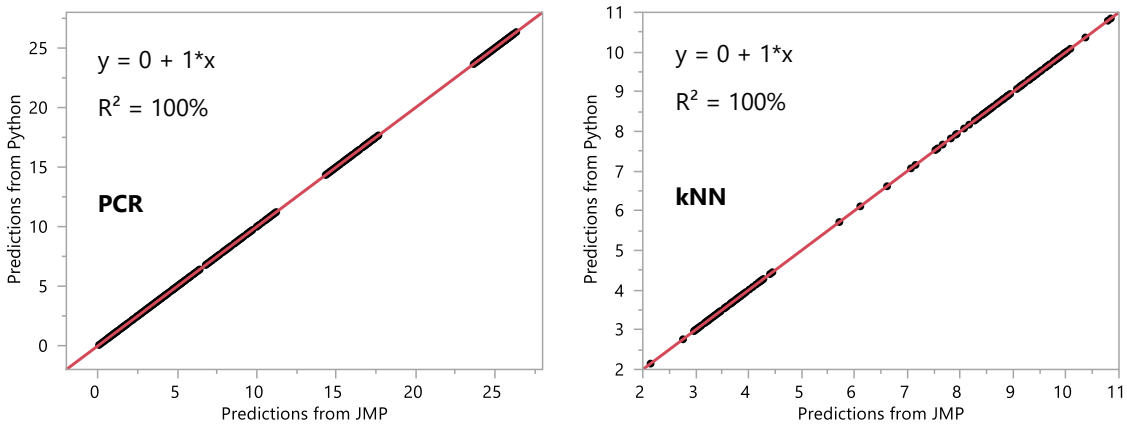

Figure 25: Comparison of the predictions of PCR and kNN from JMP and Python.

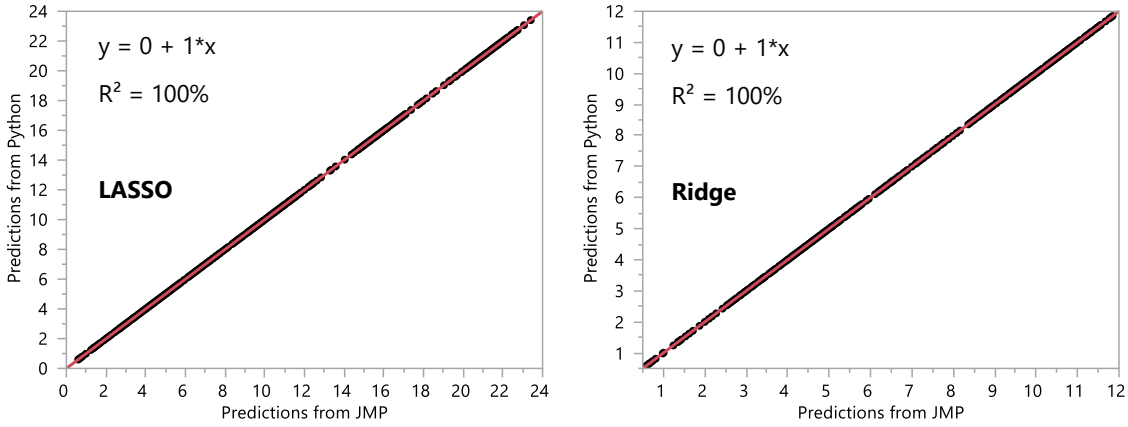

Figure 26: Comparison of the predictions of LASSO and Ridge from JMP and Python.

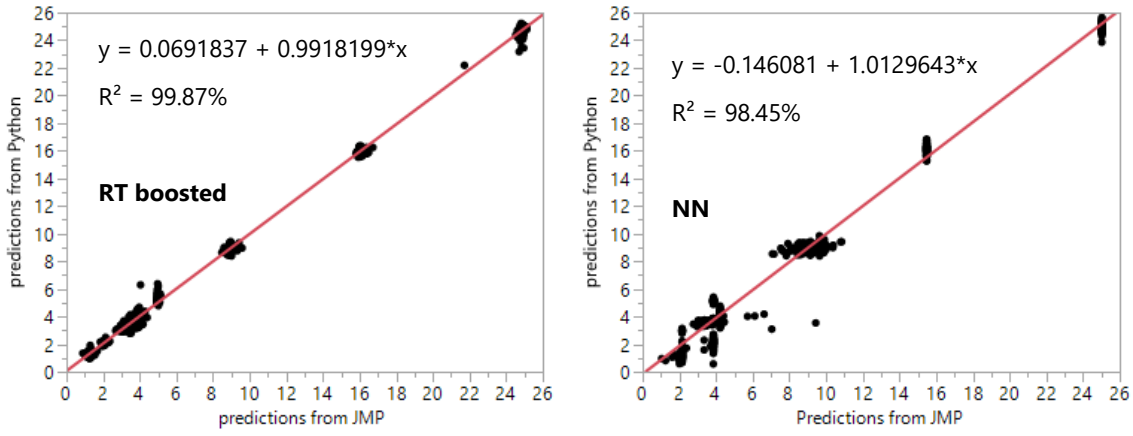

Figure 27: Comparison of the predictions of RT boosted and NN from JMP and Python.

CHAPTER SIX
CONCLUSIONS AND RECOMMENDATIONS

A software program with automated data fusion, quality assessment, variable preselection, and predictive modeling capabilities was developed in this data science research study. Improving the quality of the data significantly improved predictive modeling capabilities across a host of common modeling techniques (e.g., MLR, PLS, NN, Boosted RT, etc.) as measured by the NRMSEP and RMSEP. In simulated datasets, data imputation reduced information loss by almost 95%, and in an actual industry dataset information loss was reduced by 91%. Overall the software program reduced the NRMSEP for predicting quality product attributes from 63% to 12%.

With the exclusion of variables with extreme high missing data rates (more than 40% of missing data in one variable) 10% of the untreated data records could be predicted. Without exclusion, not a single untreated datapoint was predictable. In contrary, the treated dataset was capable of predicting 100% of the records with a small NRMSEP for dependent variables.

Improved predictive analytics key quality attributes by improved data quality may avoid mismanufacture of product, reduce rework, reduce scrap and promote process optimization.

A significant contribution of this research for the sustainable biomaterials industries compared to previous studies was that in prior studies one imputation method was sued for all variables of a data set (Zeng *et al.*, 2016; Tian *et al.*, 2018). In this study every predictor in a dataset could potentially have a different imputation method. Different imputation methods (indicator, median, mean, random, and LOCF), could reduce information loss and enhance the robustness of predictive modeling efforts. This may also make predictive analytics and data mining outcomes more accurate and useful as a business instrument for the sustainable biomaterials industries.

Data collected by the biomaterials industries may not be usable with data quality improvement. Improved data quality may also help these industries achieve ‘Industry 4.0.’

The software program developed in this thesis is able to work on a broad variety of production lines. However, further research is recommended on studying more imputation methods. More research on outlier assessment for non-normal data is required. The LASSO pre-selection cut-off value was not tested in this study as well as the variable selection (% missingness for exclusion). Further study is required with the cut-off value to assess the robustness of predictions.

LIST OF REFERENCES

Addinsoft (2018) *Partial Least Squares PLS regression in Excel | XLSTAT*. Available at: https://help.xlstat.com/customer/en/portal/articles/2062244-running-a-partial-least-squares-regression-with-xlstat?b_id=9283 (Accessed: 26 March 2019).

Akinwande, M. O., Dikko, H. G. and Samson, A. (2015) 'Variance Inflation Factor: As a Condition for the Inclusion of Suppressor Variable(s) in Regression Analysis', *Open Journal of Statistics*, 05(07), pp. 754–767. doi: 10.4236/ojs.2015.57075.

Allison, P. D. (2000) 'Multiple Imputation for Missing Data A Cautionary Tale', *Sociologiacl Methodes and Reseach*, 28, p. pp.301-309. Available at: papers2://publication/uuid/D9CA75EE-7076-44F1-A831-3B504D380545.

American Forest & Paper Association (2018) *Economic Impact*. Available at: https://www.afandpa.org/advocacy/economic-impact (Accessed: 1 July 2019).

Anderson, D. R. (2008) *Model based inference in the life sciences : a primer on evidence*. Springer.

André, N. *et al.* (2008) 'Prediction of internal bond strength in a medium density fiberboard process using multivariate statistical methods and variable selection', *Wood Science and Technology*, 42(7), pp. 521–534. doi: 10.1007/s00226-008-0204-7.

André, N. and Young, T. M. (2013) 'Real-time process modeling of particleboard manufacture using variable selection and regression methods ensemble', *European Journal of Wood and Wood Products*, 71(3), pp. 361–370. doi: 10.1007/s00107-013-0689-0.

Arcidiacono, G. and Pieroni, A. (2018) 'The Revolution Lean Six Sigma 4.0', *International Journal on Advanced Science, Engineering and Information Technology*. doi: 10.18517/ijaseit.8.1.4593.

Ardagna, D. *et al.* (2018) 'Context-aware data quality assessment for big data', *Future Generation Computer Systems*. Elsevier B.V., 89, pp. 548–562. doi: 10.1016/j.future.2018.07.014.

Augustian, I. R. *et al.* (2018) 'Economic Analysis on Real Time Data Fusion

Using Kibana & ElasticSearch', *Journal of Network Communications and Emerging Technologies (JNCET) www.jncet.org*, 8(10). Available at: www.jncet.org (Accessed: 21 March 2019).

Bain, A. (1873) *Mind and body. The theories of their relation*. D. Appleton and company. Available at: https://archive.org/details/mindbodytheories00bain/page/n6 (Accessed: 3 July 2019).

Barton, D. and Court, D. (no date) *Making Advanced Analytics Work For You A practical guide to capitalizing on big data*. Available at: http://www.buyukverienstitusu.com/s/1870/i/Making_Advanced_Analytics_Work_For_You.pdf (Accessed: 3 July 2019).

Batini, C. *et al.* (2015) 'From Data Quality to Big Data Quality', *Journal of Database Management*, 26(1), pp. 60–82. doi: 10.4018/JDM.2015010103.

Becker, D., King, T. D. and McMullen, B. (2015) 'Big data, big data quality problem', in *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*, pp. 2644–2653. doi: 10.1109/BigData.2015.7364064.

Bhadeshia, H. K. D. H. (1999) *Revie w Neural Networks in Materials Science*, *ISIJ International*. Available at: https://www.phase-trans.msm.cam.ac.uk/abstracts/neural.review.pdf (Accessed: 21 May 2019).

Bibby, L. and Dehe, B. (2018) 'Defining and assessing industry 4.0 maturity levels–case of the defence sector', *Production Planning and Control*. doi: 10.1080/09537287.2018.1503355.

Bronshtein, A. (2017) *Train/Test Split and Cross Validation in Python*. Available at: https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6 (Accessed: 21 May 2019).

Brown, B., Chui, M. and Manyika, J. (2011) 'Are you ready for the era of "big data"?', *McKinsey & Company*. Available at: https://www.mckinsey.com/business-functions/strategy-and-corporate-finance/our-insights/are-you-ready-for-the-era-of-big-data (Accessed: 3 July 2019).

Brownlee, J. (2018) *A Gentle Introduction to k-fold Cross-Validation*.

Available at: https://machinelearningmastery.com/k-fold-cross-validation/ (Accessed: 28 March 2019).

Cai, L. and Zhu, Y. (2015) 'The Challenges of Data Quality and Data Quality Assessment in the Big Data Era', *Data Science Journal*, 14(0), p. 2. doi: 10.5334/dsj-2015-002.

Castanedo, F. (2013) 'A review of data fusion techniques', *The Scientific World Journal*, p. 19. doi: 10.1155/2013/704504.

Chen, L. *et al.* (2009) 'Multiple Classifier Integration for the Prediction of Protein Structural Classes', *Wiley InterScience*, 30, p. 2248+2254. doi: 10.1002/jcc.21230.

Cordeiro, G. O., Deschamps, F. and Pinheiro de Lima, E. (2017) 'Managing a Big Data / Analytics project : a systematic literature review', (July), p. 5.

Darlington, R. B. and Hayes, A. F. (2017) *Regression analysis and linear models : concepts, applications, and implementation*.

Deming, W. A. and Shewhart, W. E. (1986) *Statistical Method from the Viewpoint of Quality Control*. Toronto: General Publishing Company. Available at: https://books.google.com/books?hl=de&lr=&id=ALGbNNMdnHkC&oi=fnd&pg=PA1&dq=deming&ots=Gt4HIKAqlN&sig=DMYm6XdWzSloH1tuvAr7af1lxq8#v=onepage&q=deming&f=false (Accessed: 3 July 2019).

Donges, N. (2018) *Pros and Cons of Neural Networks – Towards Data Science*. Available at: https://towardsdatascience.com/hype-disadvantages-of-neural-networks-6af04904ba5b (Accessed: 26 March 2019).

Dumbill, E. (2013) 'Making Sense of Big Data', *Big Data*. Mary Ann Liebert, Inc. 140 Huguenot Street, 3rd Floor New Rochelle, NY 10801 USA , 1(1), pp. 1–2. doi: 10.1089/big.2012.1503.

Dunham and Associates (2017) *Economic impact study - U.S. wood industry*. Available at: https://www.google.com/search?biw=1536&bih=754&ei=T0QdXc6aB4bm_QaHtrjgBA&q=Dunham+and+Associates+2017+wood+economic+impact&oq=Dunham+and+Associates+2017+wood+economic+impact&gs_l=psy-

ab.3..33i160.3794.11689..11834...1.0..0.217.2008.19j3j1.....0....1..gws (Accessed: 3 July 2019).

ebc Inc. (2017) *Cross-Validation Strategies | ebc*. Available at: http://www.ebc.cat/2017/01/31/cross-validation-strategies/ (Accessed: 28 March 2019).

Efron, B. (1979) 'Bootstrap Methods: Another Look at the Jackknife', *The Annals of Statistics*. Institute of Mathematical Statistics, 7(1), pp. 1–26. doi: 10.1214/aos/1176344552.

Elith, J., Leathwick, J. R. and Hastie, T. (2008) 'A working guide to boosted regression trees', *Journal of Animal Ecology*. John Wiley & Sons, Ltd (10.1111), 77(4), pp. 802–813. doi: 10.1111/j.1365-2656.2008.01390.x.

Flom, P. (2018) *The Disadvantages of Linear Regression*. Available at: https://sciencing.com/disadvantages-linear-regression-8562780.html (Accessed: 22 March 2019).

Francisco, M. M. C. *et al.* (2017) 'Total Data Quality Management and Total Information Quality Management Applied to Costumer Relationship Management', *Proceedings of the 9th International Conference on Information Management and Engineering - ICIME 2017*, (1), pp. 40–45. doi: 10.1145/3149572.3149575.

Gandomi, A. and Haider, M. (2015) 'Beyond the hype: Big data concepts, methods, and analytics', *International Journal of Information Management*. Elsevier Ltd, 35(2), pp. 137–144. doi: 10.1016/j.ijinfomgt.2014.10.007.

Gartner Inc. (2019) *Predictive Modeling*. Available at: https://www.gartner.com/it-glossary/predictive-modeling (Accessed: 21 March 2019).

Gelman, A. (2011) *Induction and deduction in Bayesian data analysis*. Available at: http://www.stat.columbia.edu/~gelman/research/unpublished/philosophy_online4.pdf (Accessed: 3 July 2019).

Goyal, P. (2018) *What are the disadvantages of a PCA?* Available at: https://www.quora.com/What-are-the-disadvantages-of-a-PCA (Accessed: 3 July

2019).

Grubbs, F. E. (1969) ‘Procedures for Detecting Outlying Observations in Samples’, *Technometrics*, 11(1), pp. 1–21. Available at: https://pdfs.semanticscholar.org/9dfa/3cb3deb2e9a18459340b5a2d3a1ea4d8d5e6.pdf (Accessed: 2 June 2019).

Gupta, D. and Rani, R. (2018) ‘A study of big data evolution and research challenges’, *Journal of Information Science*, 45(3), pp. 322–340. doi: 10.1177/0165551518789880.

Hazen, B. T. *et al.* (2017) ‘Toward understanding outcomes associated with data quality improvement’, *International Journal of Production Economics*. Elsevier Ltd, 193(August), pp. 737–747. doi: 10.1016/j.ijpe.2017.08.027.

Hoerl, A. E. and Kennard, R. W. (1970) ‘Ridge Regression: Biased Estimation for Nonorthogonal Problems’, 12(1), pp. 55–67. Available at: https://www.math.arizona.edu/~hzhang/math574m/Read/RidgeRegressionBiasedEstimationForNonorthogonalProblems.pdf (Accessed: 2 July 2019).

Iyengar, S. S., Sastry, S. and Balakrishnan, N. (2003) ‘Foundations of data fusion for automation’, *IEEE Instrumentation & Measurement Magazine*. doi: 1094-6969/03/.

Jabbar, S. *et al.* (2018) ‘A Methodology of Real-Time Data Fusion for Localized Big Data Analytics’, *IEEE Access*, 6, pp. 24510–24520. doi: 10.1109/ACCESS.2018.2820176.

Jodlowska, E. (2018) *By the numbers: Python community trends in 2017/2018 | Opensource.com*. Available at: https://opensource.com/article/18/5/numbers-python-community-trends (Accessed: 21 March 2019).

Kelley, R. (2017) *Want a Robust Model? Try These Validation Strategies [Infographic]*. Available at: https://blog.dataiku.com/want-a-robust-model-try-these-validation-strategies (Accessed: 28 March 2019).

Kenton, W. (2019) *Multiple Linear Regression – MLR Definition*. Available at: https://www.investopedia.com/terms/m/mlr.asp (Accessed: 22 March 2019).

Khoury, M. J. and Ioannidis, J. P. A. (2014) ‘Big data meets public health’, *Science*, 346(6213), pp. 1054–1055. doi: 10.1126/science.aaa2709.

Kuhn, M. and Johnson, K. (2013) *Applied Predictive Modeling*. New York, NY: Springer New York. doi: 10.1007/978-1-4614-6849-3.

Lachenbruch, P. A. and Ray Mickey, M. (1968) ‘Estimation of error rates in discriminant analysis’, *Technometrics*, 10(1), pp. 1–11. Available at: https://www.jstor.org/stable/pdf/1266219.pdf?refreqid=excelsior%3A38b4df0fd754fe6687444af8f968f1a2 (Accessed: 2 June 2019).

Laney, D. (2001) *3D Data Management: Controlling Data Volume, Velocity, and Variety*. Available at: https://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf (Accessed: 3 July 2019).

Li, C. and Wang, B. (2014) *Principal Components Analysis*. Available at: http://www.ccs.neu.edu/home/vip/teach/MLcourse/5_features_dimensions/lecture_notes/PCA/PCA.pdf (Accessed: 27 March 2019).

Liao, Y. *et al.* (2017) ‘Past, present and future of Industry 4.0 - a systematic literature review and research agenda proposal’, *International Journal of Production Research*. Taylor & Francis, 55(12), pp. 3609–3629. doi: 10.1080/00207543.2017.1308576.

Liu, J. *et al.* (2016) ‘Rethinking big data: A review on the data quality and usage issues’, *ISPRS Journal of Photogrammetry and Remote Sensing*. International Society for Photogrammetry and Remote Sensing, Inc. (ISPRS), 115, pp. 134–142. doi: 10.1016/j.isprsjprs.2015.11.006.

Liu, R. X. *et al.* (2003) ‘Principal component regression analysis with spss’, *Computer Methods and Programs in Biomedicine*. Elsevier, 71(2), pp. 141–147. doi: 10.1016/S0169-2607(02)00058-5.

Liu, T., Song, S. and Duan, G. (2016) ‘The application and management of big data in quality engineering’, in *Communications in Computer and Information Science*, pp. 624–631. doi: 10.1007/978-981-10-2669-0_66.

Lorber, A., Wangen, L. E. and Kowalski, B. R. (1987) ‘A theoretical

foundation for the PLS algorithm', *Journal of Chemometrics*, 1(1), pp. 19–31. doi: 10.1002/cem.1180010105.

Lund, H. (2017) *SIX BENEFITS OF DATA INTEGRATION*. Available at: https://www.rapidionline.com/blog/6-benefits-erp-crm-data-integration (Accessed: 19 March 2019).

MathWorks Inc. (2019) *Predictive Modeling - Time-Series Regression, Linear Regression Models*. Available at: https://www.mathworks.com/discovery/predictive-modeling.html (Accessed: 21 March 2019).

Merriam-Webster (2019) *time lag*. Available at: https://www.merriam-webster.com/dictionary/time lag (Accessed: 19 March 2019).

Micic, N. *et al.* (2018) 'Towards a Data Quality Framework for Heterogeneous Data', in *Proceedings - 2017 IEEE International Conference on Internet of Things, IEEE Green Computing and Communications, IEEE Cyber, Physical and Social Computing, IEEE Smart Data, iThings-GreenCom-CPSCom-SmartData 2017*. doi: 10.1109/iThings-GreenCom-CPSCom-SmartData.2017.28.

MicroStrategy (2019) *Predictive Modeling: The Only Guide You Need*. Available at: https://www.microstrategy.com/us/resources/introductory-guides/predictive-modeling-the-only-guide-you-need (Accessed: 21 March 2019).

MongoDB Inc. (2019) *Informationen zu NoSQL-Datenbanken*. Available at: https://www.mongodb.com/nosql-explained?lang=de-de (Accessed: 18 March 2019).

Montgomery, D. C. and Peck, E. A. (1982) *Introduction to linear regression analysis*. Wiley.

Moore, G. (2012) *Thoughts from the week #1*, *Twitter*. Available at: https://twitter.com/geoffreyamoore/status/234839087566163968?lang=de (Accessed: 4 July 2019).

Morgan, D. (2018) *Intelligent enterprises for the 4th industrial revolution*. Available at: https://www.avanade.com/en/blogs/avanade-insights/avanade-spotlight/intelligent-enterprises-for-the-4th-industrial-revolution (Accessed: 20

March 2019).

Mrugalska, B. and Wyrwicka, M. K. (2017) ‘Towards Lean Production in Industry 4.0’, in *Procedia Engineering*. doi: 10.1016/j.proeng.2017.03.135.

Nakagawa, S. and Freckleton, R. P. (2008) ‘Missing inaction: the dangers of ignoring missing data’, *Trends in Ecology and Evolution*, 23(11), pp. 592–596. doi: 10.1016/j.tree.2008.06.014.

Oxley, M. E. and Thorsen, S. N. (2006) *Fusion or Integration: What’s the difference?* Available at: https://afc2isrc.af.mil/warfighter (Accessed: 21 March 2019).

Özel, T. and Karpat, Y. (2005) ‘Predictive modeling of surface roughness and tool wear in hard turning using regression and neural networks’, *International Journal of Machine Tools and Manufacture*. Pergamon, 45(4–5), pp. 467–479. doi: 10.1016/J.IJMACHTOOLS.2004.09.007.

Panetto, H. and Molina, A. (2008) ‘Enterprise integration and interoperability in manufacturing systems: Trends and issues’, *Computers in Industry*. Elsevier, 59(7), pp. 641–646. doi: 10.1016/J.COMPIND.2007.12.010.

Pearson, K. (1901) ‘LIII. On lines and planes of closest fit to systems of points in space’, *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*. Taylor & Francis Group, 2(11), pp. 559–572. doi: 10.1080/14786440109462720.

Pearson, K. (1930) *Life, Letters and Labours of Francis Galton*. III. University of London, Cambridge. Available at: http://galton.org/pearson/ocr/vol3b.pdf (Accessed: 3 July 2019).

Peres, F. A. P. and Fogliatto, F. S. (2018) ‘Variable selection methods in multivariate statistical process control: A systematic literature review’, *Computers and Industrial Engineering*, 115, pp. 603–619. doi: 10.1016/j.cie.2017.12.006.

Petkov, A. (2018) *Here are the best programming languages to learn in 2018*. Available at: https://medium.freecodecamp.org/best-programming-languages-to-learn-in-2018-ultimate-guide-bfc93e615b35 (Accessed: 21 March 2019).

Pipino, L. L. *et al.* (2002) *Data Quality Assessment*. Available at: http://web.mit.edu/tdqm/www/tdqmpub/PipinoLeeWangCACMApr02.pdf (Accessed: 2 June 2019).

Python Software Foundation (2019) *Legal Statements | Python Software Foundation*. Available at: https://www.python.org/about/legal/ (Accessed: 21 March 2019).

Qi, Q. and Tao, F. (2018) 'Digital Twin and Big Data Towards Smart Manufacturing and Industry 4.0: 360 Degree Comparison', *IEEE Access*, 6, pp. 3585–3593. doi: 10.1109/ACCESS.2018.2793265.

Quinlan, J. R. (1986) *Induction of Decision Trees*, *Machine Learning*. Available at: https://link.springer.com/content/pdf/10.1007/BF00116251.pdf (Accessed: 21 May 2019).

Rao, D., Gudivada, V. N. and Raghavan, V. V. (2015) 'Data quality issues in big data', in *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*, pp. 2654–2660. doi: 10.1109/BigData.2015.7364065.

Raschka, S. (2018) *Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning*. Available at: https://arxiv.org/pdf/1811.12808.pdf (Accessed: 27 March 2019).

Robinson, D. (2017) *The Incredible Growth of Python*. Available at: https://stackoverflow.blog/2017/09/06/incredible-growth-python/ (Accessed: 21 March 2019).

Roblek, V., Meško, M. and Krapež, A. (2016) 'A Complex View of Industry 4.0', *SAGE Open*. doi: 10.1177/2158244016653987.

Saidulu, D. (2017) 'Machine Learning and Statistical Approaches for Big Data : Issues , Challenges and Research Directions', *International Journal of Applied Engineering Research*, 12(21), pp. 11691–11699.

Saptashwa, B. (2018) *Ridge and Lasso Regression: A Complete Guide with Python Scikit-Learn*. Available at: https://towardsdatascience.com/ridge-and-lasso-regression-a-complete-guide-with-python-scikit-learn-e20e34bcbf0b (Accessed: 22 March 2019).

SAS Institution Inc. (2019a) *Neural Networks - What they are and why they matter*. Available at: https://www.sas.com/en_us/insights/analytics/neural-networks.html (Accessed: 26 March 2019).

SAS Institution Inc. (2019b) *Predictive Analytics - What it is and why it matters*. Available at: https://www.sas.com/en_us/insights/analytics/predictive-analytics.html#dmtechnical (Accessed: 22 March 2019).

Schadt, E. E. *et al.* (2011) 'Cloud and heterogeneous computing solutions exist today for the emerging big data problems in biology', *Nature Reviews Genetics*. Nature Publishing Group, 12(3), pp. 224–224. doi: 10.1038/nrg2857-c2.

Seif, G. (2018) *5 Types of Regression and their properties*. Available at: https://towardsdatascience.com/5-types-of-regression-and-their-properties-c5e1fa12d55e (Accessed: 22 March 2019).

Shlens, J. (2005) *A Tutorial on Principal Component Analysis*. Available at: https://www.cs.cmu.edu/~elaw/papers/pca.pdf (Accessed: 2 July 2019).

skymind (2018) *A Beginner's Guide to Neural Networks and Deep Learning | Skymind*. Available at: https://skymind.ai/wiki/neural-network (Accessed: 26 March 2019).

Stock, T. and Seliger, G. (2016) 'Opportunities of Sustainable Manufacturing in Industry 4.0', in *Procedia CIRP*. doi: 10.1016/j.procir.2016.01.129.

Sutton, C. D. (2005) 'Classification and Regression Trees, Bagging, and Boosting', *Handbook of Statistics*. Elsevier, 24, pp. 303–329. doi: 10.1016/S0169-7161(04)24011-1.

Taguchi, G. *et al.* (2005) *Taguchi's quality engineering handbook*. John Wiley & Sons.

Tao, F. *et al.* (2018) 'Data-driven smart manufacturing', *Journal of Manufacturing Systems*. Elsevier, 48, pp. 157–169. doi: 10.1016/J.JMSY.2018.01.006.

Theorin, A. *et al.* (2015) 'An Event-Driven Manufacturing Information System Architecture', *IFAC-PapersOnLine*. Elsevier Ltd., 48(3), pp. 547–554. doi:

10.1016/j.ifacol.2015.06.138.

Thomas, B. and Jacob, G. (2016) ‘Big Data –The Change of Datacenter Concept and Challenges’, *Iarjset*, 3(7), pp. 261–263. doi: 10.17148/IARJSET.2016.3754.

Tian, N. P. *et al.* (2018) ‘Improved Predictive Modeling Using Bayesian Additive Regression Trees (BART) For Wood Composite Products’, *The International Journal of Engineering and Science (IJES)*, 7(5), pp. 66–75. doi: 10.9790/1813-0705036675.

Tibishirani, R. (1996) ‘Regression Shrinkage and Selection via the Lasso’, *Journal of the Royal Statistical Society*, 58(1), pp. 267–288. Available at: http://statweb.stanford.edu/~tibs/lasso/lasso.pdf (Accessed: 3 July 2019).

Tufts, C. (2015) *Classification Model Pros and Cons*. Available at: https://github.com/ctufts/Cheat_Sheets/wiki/Classification-Model-Pros-and-Cons (Accessed: 27 March 2019).

U.S. Department of Energy (2018) *Forest Products Industry Profile*. Available at: https://www.energy.gov/eere/amo/forest-products-industry-profile (Accessed: 20 March 2019).

U.S. Department of Transportation (2017) *Why Integrate Data?* Available at: https://www.fhwa.dot.gov/asset/dataintegration/if10019/dip04.cfm (Accessed: 19 March 2019).

U.S. Environmental Protection Agency (2017) *Greenhouse Gases Equivalencies Calculator - Calculations and References*. Available at: http://link.sprhttps//www.epa.gov/energy/greenhouse-gases-equivalencies-calculator-calculations-and-referencesinger.com/10.1007/978-3-642-21726-5_2 (Accessed: 3 July 2019).

U.S Chamber of Commerce (2018) *Comments on Clean Water Act Coverage of “Discharges of Pollutants” via a Direct Hydrologic Connection to Surface Water*. Available at: https://www.uschamber.com/comment/comments-clean-water-act-coverage-of-discharges-of-pollutants-direct-hydrologic-connection (Accessed: 20 March 2019).

Vandone, A., Baraldo, S. and Valente, A. (2018) ‘Multisensor Data Fusion for Additive Manufacturing Process Control’, *IEEE Robotics and Automation Letters*, 3(4), pp. 3279–3284. doi: 10.1109/LRA.2018.2851792.

w3schools (2019) *Introduction to SQL*. Available at: https://www.w3schools.com/sql/sql_intro.asp (Accessed: 18 March 2019).

Witten, I. H., Frank, E. and Hall, M. A. (2011) ‘Ensemble Learning’, *Data Mining: Practical Machine Learning Tools and Techniques*, pp. 351–373. doi: 10.1016/B978-0-12-374856-0.00008-0.

Wold, S., Esbensen, K. and Geladi, P. (1987) ‘Principal component analysis’, *Chemometrics and Intelligent Laboratory Systems*. Elsevier, 2(1–3), pp. 37–52. doi: 10.1016/0169-7439(87)80084-9.

Young, T. M. *et al.* (2008) ‘A comparison of multiple linear regression and quantile regression for modeling the internal bond of medium density fiberboard’, *Forest Products Journal*. doi: 10.1080/08982112.2014.968667.

Young, T. M. *et al.* (2014) ‘Predicting Key Reliability Response with Limited Response Data’, 26, pp. 223–232. doi: 10.1080/08982112.2013.807930.

Zeng, Y. (2011) ‘A Study of Missing Data Imputation and Predictive Modeling of Strength Properties of Wood Composites’, *Universitry of Tennessee*. doi: 10.1002/eji.1830221036.

Zeng, Y. *et al.* (2016) ‘A Study of Missing Data Imputation in Predictive Modeling of a Wood-Composite Manufacturing Process.’, *Journal of Quality Technology*, 48(3), pp. 284–296. doi: 10.1080/00224065.2016.11918167.

Zhang, Z. (2016) ‘Missing data imputation: focusing on single imputation’, *Hemodialysis International, Journal of Translational Medicine*, 4(1), p. 9. doi: 10.3978/j.issn.2305-5839.2015.12.38.

Zhao, S. (2016) *Manual vs. automated data validation | Experian*. Available at: https://www.edq.com/blog/manual-vs.-automated-data-validation/ (Accessed: 28 March 2019).

Zheng, A. (2015) *Evaluating Machine Learning Models*. Available at: https://www.oreilly.com/ideas/evaluating-machine-learning-models/page/4/offline-

evaluation-mechanisms-hold-out-validation-cross-validation-and-bootstrapping (Accessed: 21 May 2019).

Zhong, R. Y. *et al.* (2017) ‘Intelligent Manufacturing in the Context of Industry 4.0: A Review’, *Engineering*. Elsevier LTD on behalf of Chinese Academy of Engineering and Higher Education Press Limited Company, 3(5), pp. 616–630. doi: 10.1016/J.ENG.2017.05.015.

APPENDIX

10-page example of the report file

Variable:	Thickness_swell_ Indicator		Median		Mean		Random		LOCF	
fold	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	2.61	<1.0%	4.55	1.0%	7.03	1.0%	10.69	2.0%	9.83	1.0%
2	3.86	<1.0%	3.35	<1.0%	5.82	<1.0%	9.53	<1.0%	11.20	1.0%
3	4.40	<1.0%	2.52	<1.0%	4.93	<1.0%	13.58	1.0%	16.12	1.0%
4	7.31	1.0%	3.58	<1.0%	3.46	<1.0%	11.04	1.0%	15.23	1.0%
5	12.37	1.0%	8.42	<1.0%	5.06	<1.0%	4.42	<1.0%	9.38	<1.0%
6	10.78	<1.0%	7.27	<1.0%	5.35	<1.0%	13.79	1.0%	7.77	<1.0%
7	3.20	<1.0%	4.41	<1.0%	6.75	<1.0%	17.25	1.0%	0.71	<1.0%
8	10.67	1.0%	7.69	1.0%	6.67	1.0%	12.70	2.0%	1.08	<1.0%
9	15.00	1.0%	10.92	1.0%	7.84	1.0%	5.71	<1.0%	7.64	1.0%
10	10.85	1.0%	7.48	1.0%	5.95	1.0%	12.61	1.0%	8.80	1.0%
Average	8.11		6.02		5.89		11.13		8.78	
Std dev	4.39		2.71		1.26		3.83		5.05	

Variable:	pressure Indicator		Median		Mean		Random		LOCF	
fold	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	210.92	3.0%	191.23	3.0%	210.23	3.0%	105.84	2.0%	787.03	11.0%
2	515.36	3.0%	311.44	2.0%	276.94	1.0%	757.51	4.0%	857.95	4.0%
3	611.67	3.0%	408.36	2.0%	286.25	1.0%	522.26	2.0%	448.99	2.0%
4	959.08	8.0%	772.80	7.0%	710.29	6.0%	703.09	6.0%	803.11	7.0%
5	899.07	5.0%	699.37	4.0%	679.00	4.0%	409.03	2.0%	819.81	4.0%
6	142.59	1.0%	254.33	1.0%	286.79	1.0%	509.51	2.0%	469.70	2.0%
7	243.31	1.0%	164.26	1.0%	210.73	1.0%	471.89	3.0%	292.85	2.0%
8	403.42	5.0%	203.16	3.0%	79.14	1.0%	344.40	4.0%	294.74	4.0%
9	335.38	2.0%	220.66	1.0%	266.64	2.0%	435.20	3.0%	118.75	1.0%
10	335.38	3.0%	220.66	2.0%	266.64	3.0%	696.85	7.0%	200.82	2.0%
Average	465.62		344.63		327.26		495.56		509.38	
Std dev	281.19		218.44		203.44		194.23		284.26	

Variable:	QC_MOR									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	5614.47	8.0%	73.37	<1.0%	1024.84	1.0%	4772.43	7.0%	202.20	<1.0%
2	3931.01	2.0%	3969.95	2.0%	3614.24	2.0%	1803.84	1.0%	4043.98	2.0%
3	4010.28	2.0%	3963.11	2.0%	3614.34	2.0%	3819.27	2.0%	74.17	<1.0%
4	5662.54	5.0%	12.96	<1.0%	1081.40	1.0%	853.54	1.0%	3969.78	3.0%
5	5715.31	3.0%	87.11	<1.0%	1145.93	1.0%	1050.72	1.0%	13.23	<1.0%
6	5706.53	2.0%	82.81	<1.0%	1135.80	<1.0%	1724.21	1.0%	129.63	<1.0%
7	3985.61	2.0%	3969.24	2.0%	3614.07	2.0%	3365.06	2.0%	24.66	<1.0%
8	4139.81	5.0%	3965.45	5.0%	3619.06	5.0%	3342.98	4.0%	5651.58	7.0%
9	5720.86	4.0%	146.80	<1.0%	1157.35	1.0%	739.97	1.0%	155.24	<1.0%
10	5709.45	6.0%	132.29	<1.0%	1142.78	1.0%	4130.31	4.0%	40.00	<1.0%
Average	5019.59		1640.31		2114.98		2560.23		1430.45	
Std dev	865.24		2002.76		1291.95		1489.98		2203.06	

Variable:	bin_temp									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	20.04	3.0%	0.04	<1.0%	0.03	<1.0%	1.04	<1.0%	0.01	<1.0%
2	20.00	1.0%	0.01	<1.0%	0.01	<1.0%	0.22	<1.0%	0.02	<1.0%
3	20.00	1.0%	0.01	<1.0%	0.01	<1.0%	0.87	<1.0%	< 0.00	<1.0%
4	20.02	2.0%	0.02	<1.0%	0.01	<1.0%	0.67	<1.0%	0.01	<1.0%
5	20.03	1.0%	0.03	<1.0%	0.02	<1.0%	0.97	<1.0%	0.03	<1.0%
6	20.01	1.0%	0.01	<1.0%	< 0.00	<1.0%	0.85	<1.0%	0.01	<1.0%
7	20.02	1.0%	0.02	<1.0%	0.01	<1.0%	0.04	<1.0%	0.03	<1.0%
8	19.99	2.0%	0.02	<1.0%	0.02	<1.0%	0.18	<1.0%	0.01	<1.0%
9	20.00	1.0%	0.01	<1.0%	0.01	<1.0%	0.63	<1.0%	0.02	<1.0%
10	19.98	2.0%	0.03	<1.0%	0.03	<1.0%	0.34	<1.0%	0.06	<1.0%
Average	20.01		0.02		0.01		0.58		0.02	
Std dev	0.02		0.01		0.01		0.36		0.02	

Variable:	Moisture_cont_bin 1									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	0.76	1.0%	0.06	<1.0%	0.06	<1.0%	0.40	1.0%	0.13	<1.0%
2	0.72	<1.0%	0.10	<1.0%	0.07	<1.0%	0.45	<1.0%	0.03	<1.0%
3	0.69	<1.0%	0.15	<1.0%	0.13	<1.0%	0.16	<1.0%	0.05	<1.0%
4	0.76	1.0%	0.12	<1.0%	0.12	<1.0%	0.30	<1.0%	0.16	<1.0%
5	0.94	1.0%	0.24	<1.0%	0.21	<1.0%	0.78	<1.0%	0.13	<1.0%
6	0.93	<1.0%	0.24	<1.0%	0.20	<1.0%	0.98	<1.0%	0.15	<1.0%
7	0.78	1.0%	0.09	<1.0%	0.09	<1.0%	0.42	<1.0%	0.03	<1.0%
8	0.77	1.0%	0.08	<1.0%	0.08	<1.0%	0.11	<1.0%	0.09	<1.0%
9	0.76	1.0%	0.08	<1.0%	0.08	<1.0%	0.50	<1.0%	0.17	<1.0%
10	0.69	1.0%	0.13	<1.0%	0.10	<1.0%	0.56	1.0%	0.25	<1.0%
Average	0.78		0.13		0.11		0.47		0.12	
Std dev	0.09		0.06		0.05		0.26		0.07	

Variable:	Moisture_cont_bin 2									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	0.71	1.0%	< 0.00	<1.0%	< 0.00	<1.0%	0.09	<1.0%	0.01	<1.0%
2	0.71	<1.0%	< 0.00	<1.0%	< 0.00	<1.0%	0.51	<1.0%	0.01	<1.0%
3	0.71	<1.0%	< 0.00	<1.0%	< 0.00	<1.0%	0.48	<1.0%	< 0.00	<1.0%
4	0.71	1.0%	< 0.00	<1.0%	< 0.00	<1.0%	0.75	1.0%	< 0.00	<1.0%
5	0.71	<1.0%	< 0.00	<1.0%	< 0.00	<1.0%	0.53	<1.0%	< 0.00	<1.0%
6	0.71	<1.0%	< 0.00	<1.0%	< 0.00	<1.0%	0.55	<1.0%	< 0.00	<1.0%
7	0.71	<1.0%	< 0.00	<1.0%	< 0.00	<1.0%	0.37	<1.0%	< 0.00	<1.0%
8	0.71	1.0%	< 0.00	<1.0%	< 0.00	<1.0%	0.17	<1.0%	< 0.00	<1.0%
9	0.71	1.0%	0.01	<1.0%	0.01	<1.0%	0.30	<1.0%	< 0.00	<1.0%
10	0.70	1.0%	0.01	<1.0%	0.01	<1.0%	0.56	1.0%	0.01	<1.0%
Average	0.71		< 0.00		< 0.00		0.43		< 0.00	
Std dev	< 0.00		< 0.00		< 0.00		0.2		< 0.00	

Variable:	bin_level1									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	4.70	7.0%	0.31	<1.0%	0.30	<1.0%	0.46	1.0%	0.44	1.0%
2	4.86	3.0%	0.20	<1.0%	0.17	<1.0%	0.52	<1.0%	0.22	<1.0%
3	5.00	2.0%	0.17	<1.0%	0.07	<1.0%	0.32	<1.0%	0.07	<1.0%
4	5.14	5.0%	0.33	<1.0%	0.27	<1.0%	0.94	1.0%	0.30	<1.0%
5	5.14	3.0%	0.34	<1.0%	0.27	<1.0%	0.39	<1.0%	0.39	<1.0%
6	4.92	2.0%	0.12	<1.0%	0.09	<1.0%	0.72	<1.0%	0.01	<1.0%
7	4.83	3.0%	0.17	<1.0%	0.13	<1.0%	0.52	<1.0%	0.33	<1.0%
8	4.75	6.0%	0.26	<1.0%	0.24	<1.0%	0.30	<1.0%	0.43	1.0%
9	4.96	3.0%	0.29	<1.0%	0.29	<1.0%	0.70	1.0%	0.40	<1.0%
10	5.17	5.0%	0.35	<1.0%	0.29	<1.0%	1.16	1.0%	0.45	<1.0%
Average	4.95		0.25		0.21		0.6		0.3	
Std dev	0.17		0.08		0.09		0.28		0.16	

Variable:	bin_level2									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	5.14	7.0%	0.14	<1.0%	0.17	<1.0%	0.78	1.0%	0.34	1.0%
2	5.06	3.0%	0.12	<1.0%	0.12	<1.0%	0.60	<1.0%	0.36	<1.0%
3	4.82	2.0%	0.32	<1.0%	0.26	<1.0%	0.46	<1.0%	0.38	<1.0%
4	4.91	4.0%	0.24	<1.0%	0.25	<1.0%	0.74	1.0%	0.46	<1.0%
5	4.97	3.0%	0.16	<1.0%	0.16	<1.0%	0.42	<1.0%	0.05	<1.0%
6	4.91	2.0%	0.23	<1.0%	0.15	<1.0%	0.67	<1.0%	0.21	<1.0%
7	5.11	3.0%	0.15	<1.0%	0.16	<1.0%	0.87	1.0%	0.20	<1.0%
8	5.11	6.0%	0.15	<1.0%	0.16	<1.0%	0.33	<1.0%	0.30	<1.0%
9	4.87	3.0%	0.27	<1.0%	0.21	<1.0%	0.66	<1.0%	0.32	<1.0%
10	4.99	5.0%	0.24	<1.0%	0.24	<1.0%	0.64	1.0%	0.15	<1.0%
Average	4.99		0.2		0.19		0.62		0.28	
Std dev	0.11		0.07		0.05		0.17		0.12	

Variable:	dryer_heat									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	199.47	288.0%	0.57	1.0%	0.60	1.0%	0.35	1.0%	0.54	1.0%
2	199.76	102.0%	0.43	<1.0%	0.43	<1.0%	1.02	1.0%	0.42	<1.0%
3	200.19	88.0%	0.21	<1.0%	0.28	<1.0%	1.27	1.0%	0.08	<1.0%
4	200.14	175.0%	0.19	<1.0%	0.25	<1.0%	0.82	1.0%	0.19	<1.0%
5	200.05	103.0%	0.06	<1.0%	0.11	<1.0%	1.22	1.0%	0.14	<1.0%
6	200.00	82.0%	0.10	<1.0%	0.10	<1.0%	1.08	<1.0%	0.50	<1.0%
7	199.95	122.0%	0.10	<1.0%	0.05	<1.0%	0.87	1.0%	0.36	<1.0%
8	199.95	249.0%	0.10	<1.0%	0.05	<1.0%	0.47	1.0%	0.15	<1.0%
9	199.99	135.0%	0.09	<1.0%	0.09	<1.0%	0.88	1.0%	0.38	<1.0%
10	200.04	197.0%	0.05	<1.0%	0.09	<1.0%	1.20	1.0%	0.38	<1.0%
Average	199.95		0.19		0.2		0.92		0.31	
Std dev	0.21		0.17		0.19		0.31		0.16	

Variable:	dryer_speed									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	29.91	43.0%	0.16	<1.0%	0.16	<1.0%	0.59	1.0%	0.21	<1.0%
2	29.92	15.0%	0.16	<1.0%	0.16	<1.0%	0.78	<1.0%	0.19	<1.0%
3	30.01	13.0%	0.03	<1.0%	0.01	<1.0%	0.63	<1.0%	0.03	<1.0%
4	30.03	26.0%	0.03	<1.0%	0.03	<1.0%	0.42	<1.0%	0.07	<1.0%
5	30.07	16.0%	0.05	<1.0%	0.06	<1.0%	0.22	<1.0%	0.04	<1.0%
6	30.05	12.0%	0.04	<1.0%	0.05	<1.0%	0.39	<1.0%	0.12	<1.0%
7	29.96	18.0%	0.10	<1.0%	0.09	<1.0%	0.85	1.0%	0.14	<1.0%
8	30.04	37.0%	0.14	<1.0%	0.14	<1.0%	0.59	1.0%	0.15	<1.0%
9	30.14	2<1.0%	0.12	<1.0%	0.15	<1.0%	0.89	1.0%	0.22	<1.0%
10	30.07	3<1.0%	0.06	<1.0%	0.07	<1.0%	0.36	<1.0%	0.17	<1.0%
Average	30.02		0.09		0.09		0.57		0.13	
Std dev	0.07		0.05		0.06		0.22		0.07	

Variable:	dryer_rotation_Speed									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	5.03	7.0%	0.03	<1.0%	0.02	<1.0%	0.19	<1.0%	0.03	<1.0%
2	5.04	3.0%	0.04	<1.0%	0.03	<1.0%	0.42	<1.0%	0.04	<1.0%
3	5.02	2.0%	0.02	<1.0%	0.02	<1.0%	0.60	<1.0%	0.02	<1.0%
4	5.01	4.0%	0.01	<1.0%	0.01	<1.0%	0.64	1.0%	0.03	<1.0%
5	5.05	3.0%	0.05	<1.0%	0.04	<1.0%	0.43	<1.0%	0.07	<1.0%
6	5.04	2.0%	0.04	<1.0%	0.04	<1.0%	0.88	<1.0%	0.07	<1.0%
7	5.02	3.0%	0.02	<1.0%	0.02	<1.0%	0.76	1.0%	0.01	<1.0%
8	5.02	6.0%	0.02	<1.0%	0.02	<1.0%	0.23	<1.0%	0.02	<1.0%
9	4.99	3.0%	0.03	<1.0%	0.03	<1.0%	0.74	1.0%	0.02	<1.0%
10	4.99	5.0%	0.03	<1.0%	0.03	<1.0%	0.35	<1.0%	0.06	<1.0%
Average	5.02		0.03		0.03		0.52		0.04	
Std dev	0.02		0.01		0.01		0.23		0.02	

Variable:	dryer_Airflow									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	5.02	7.0%	0.03	<1.0%	0.03	<1.0%	0.37	1.0%	0.03	<1.0%
2	4.99	3.0%	0.06	<1.0%	0.06	<1.0%	0.64	<1.0%	0.05	<1.0%
3	4.97	2.0%	0.05	<1.0%	0.06	<1.0%	0.61	<1.0%	0.08	<1.0%
4	5.00	4.0%	< 0.00	<1.0%	< 0.00	<1.0%	0.39	<1.0%	0.05	<1.0%
5	4.99	3.0%	0.01	<1.0%	0.01	<1.0%	0.51	<1.0%	0.01	<1.0%
6	4.99	2.0%	0.01	<1.0%	0.01	<1.0%	0.88	<1.0%	0.02	<1.0%
7	5.01	3.0%	0.01	<1.0%	0.01	<1.0%	0.65	<1.0%	0.03	<1.0%
8	5.00	6.0%	0.02	<1.0%	0.02	<1.0%	0.41	1.0%	< 0.00	<1.0%
9	4.99	3.0%	0.01	<1.0%	0.01	<1.0%	0.39	<1.0%	0.04	<1.0%
10	5.02	5.0%	0.02	<1.0%	0.02	<1.0%	0.63	1.0%	0.03	<1.0%
Average	5		0.02		0.02		0.55		0.03	
Std dev	0.02		0.02		0.02		0.16		0.02	

Variable:	dyer_Weight									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	251.59	4.0%	73.23	1.0%	69.23	1.0%	37.02	1.0%	97.65	1.0%
2	185.78	1.0%	37.69	<1.0%	50.53	<1.0%	62.43	<1.0%	36.66	<1.0%
3	208.96	1.0%	13.96	<1.0%	25.82	<1.0%	79.33	<1.0%	42.13	<1.0%
4	212.42	2.0%	10.47	<1.0%	20.18	<1.0%	45.11	<1.0%	54.86	<1.0%
5	241.53	1.0%	47.31	<1.0%	39.07	<1.0%	32.65	<1.0%	24.64	<1.0%
6	295.83	1.0%	98.48	<1.0%	84.32	<1.0%	6.60	<1.0%	93.80	<1.0%
7	272.58	2.0%	83.96	1.0%	68.35	<1.0%	132.38	1.0%	121.9	1.0%
8	193.32	2.0%	37.86	<1.0%	53.41	1.0%	52.69	1.0%	46.28	1.0%
9	165.44	1.0%	56.95	<1.0%	74.09	1.0%	144.26	1.0%	117.9	1.0%
10	264.35	3.0%	89.82	1.0%	85.98	1.0%	17.05	<1.0%	101.8	1.0%
Average	229.18		54.97		57.1		60.95		73.78	
Std dev	42.35		30.91		23.24		45.89		36.43	

Variable:	L1_weight									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	59.74	1.0%	16.27	<1.0%	23.62	<1.0%	43.83	1.0%	43.18	1.0%
2	82.47	<1.0%	20.70	<1.0%	20.36	<1.0%	41.02	<1.0%	28.65	<1.0%
3	76.38	<1.0%	30.57	<1.0%	31.90	<1.0%	43.11	<1.0%	31.89	<1.0%
4	75.59	1.0%	30.10	<1.0%	31.60	<1.0%	43.92	<1.0%	13.73	<1.0%
5	88.28	<1.0%	29.95	<1.0%	15.49	<1.0%	30.04	<1.0%	0.85	<1.0%
6	76.00	<1.0%	16.12	<1.0%	3.74	<1.0%	22.43	<1.0%	16.30	<1.0%
7	68.35	<1.0%	11.50	<1.0%	15.92	<1.0%	18.69	<1.0%	30.05	<1.0%
8	59.58	1.0%	16.42	<1.0%	23.81	<1.0%	32.71	<1.0%	11.51	<1.0%
9	93.95	1.0%	32.48	<1.0%	32.16	<1.0%	21.25	<1.0%	56.82	<1.0%
10	119.75	1.0%	59.88	1.0%	49.74	<1.0%	36.49	<1.0%	55.61	1.0%
Average	80.01		26.4		24.83		33.35		28.86	
Std dev	17.83		14		12.49		9.87		18.81	

Variable:	L1_resin									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	6.82	1<1.0%	0.83	1.0%	1.28	2.0%	2.20	3.0%	2.03	3.0%
2	6.08	3.0%	2.06	1.0%	2.79	1.0%	3.10	2.0%	4.34	2.0%
3	7.58	3.0%	2.97	1.0%	3.05	1.0%	4.86	2.0%	1.42	1.0%
4	11.03	1<1.0%	5.06	4.0%	4.20	4.0%	2.89	3.0%	1.98	2.0%
5	9.50	5.0%	3.72	2.0%	3.46	2.0%	2.74	1.0%	5.99	3.0%
6	5.98	2.0%	1.62	1.0%	1.93	1.0%	3.97	2.0%	0.09	<1.0%
7	9.41	6.0%	3.65	2.0%	3.39	2.0%	3.76	2.0%	4.37	3.0%
8	9.99	12.0%	4.30	5.0%	3.38	4.0%	4.49	6.0%	5.61	7.0%
9	6.87	5.0%	0.78	1.0%	1.20	1.0%	4.67	3.0%	3.31	2.0%
10	8.29	8.0%	2.35	2.0%	2.07	2.0%	2.02	2.0%	5.12	5.0%
Average	8.15		2.73		2.67		3.47		3.43	
Std dev	1.76		1.45		1.01		1.03		1.98	

Variable:	L1_hardner									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	1.51	2.0%	0.59	1.0%	0.61	1.0%	0.58	1.0%	0.26	<1.0%
2	2.00	1.0%	0.82	<1.0%	0.56	<1.0%	0.20	<1.0%	0.56	<1.0%
3	2.21	1.0%	1.04	1.0%	0.83	<1.0%	1.10	1.0%	0.90	<1.0%
4	1.88	2.0%	0.66	1.0%	0.67	1.0%	0.68	1.0%	0.85	1.0%
5	1.91	1.0%	0.70	<1.0%	0.70	<1.0%	0.65	<1.0%	0.25	<1.0%
6	1.82	1.0%	0.81	<1.0%	0.81	<1.0%	0.53	<1.0%	1.15	1.0%
7	1.01	1.0%	0.53	<1.0%	0.68	<1.0%	0.75	1.0%	0.33	<1.0%
8	1.35	2.0%	0.23	<1.0%	0.30	<1.0%	0.67	1.0%	0.03	<1.0%
9	1.33	1.0%	0.26	<1.0%	0.33	<1.0%	0.76	1.0%	0.26	<1.0%
10	1.33	1.0%	0.26	<1.0%	0.33	<1.0%	1.20	1.0%	0.42	<1.0%
Average	1.64		0.59		0.58		0.71		0.5	
Std dev	0.38		0.27		0.2		0.28		0.36	

Variable:	L1_melamine									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	4.95	7.0%	1.28	2.0%	1.26	2.0%	0.93	1.0%	1.73	3.0%
2	3.58	2.0%	0.93	1.0%	1.29	1.0%	0.94	1.0%	2.64	1.0%
3	3.06	1.0%	1.62	1.0%	2.08	1.0%	2.24	1.0%	3.45	2.0%
4	3.61	3.0%	1.48	1.0%	1.78	2.0%	3.21	3.0%	0.82	1.0%
5	4.50	2.0%	0.90	1.0%	0.15	<1.0%	1.82	1.0%	1.03	1.0%
6	6.05	3.0%	2.68	1.0%	2.07	1.0%	1.89	1.0%	2.05	1.0%
7	7.18	4.0%	3.58	2.0%	3.12	2.0%	4.47	3.0%	3.44	2.0%
8	5.60	7.0%	1.94	2.0%	1.95	2.0%	3.98	5.0%	1.99	3.0%
9	4.91	3.0%	1.23	1.0%	1.21	1.0%	2.62	2.0%	0.67	1.0%
10	4.55	5.0%	1.80	2.0%	1.85	2.0%	2.83	3.0%	3.59	4.0%
Average	4.8		1.74		1.68		2.49		2.14	
Std dev	1.25		0.83		0.77		1.18		1.11	

Variable:	L1_Moisture_cont									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	0.10	<1.0%	< 0.00	<1.0%	< 0.00	<1.0%	0.33	1.0%	< 0.00	<1.0%
2	0.10	<1.0%	< 0.00	<1.0%	< 0.00	<1.0%	0.62	<1.0%	< 0.00	<1.0%
3	0.10	<1.0%	< 0.00	<1.0%	< 0.00	<1.0%	0.59	<1.0%	< 0.00	<1.0%
4	0.10	<1.0%	< 0.00	<1.0%	< 0.00	<1.0%	0.68	1.0%	< 0.00	<1.0%
5	0.10	<1.0%	< 0.00	<1.0%	< 0.00	<1.0%	0.58	<1.0%	< 0.00	<1.0%
6	0.10	<1.0%	< 0.00	<1.0%	< 0.00	<1.0%	0.49	<1.0%	< 0.00	<1.0%
7	0.11	<1.0%	0.01	<1.0%	0.01	<1.0%	0.30	<1.0%	0.01	<1.0%
8	0.11	<1.0%	0.01	<1.0%	0.01	<1.0%	0.23	<1.0%	0.01	<1.0%
9	0.10	<1.0%	< 0.00	<1.0%	< 0.00	<1.0%	0.54	<1.0%	0.01	<1.0%
10	0.10	<1.0%	< 0.00	<1.0%	< 0.00	<1.0%	0.53	1.0%	< 0.00	<1.0%
Average	0.1		< 0.00		< 0.00		0.49		< 0.00	
Std dev	< 0.00		< 0.00		< 0.00		0.15		< 0.00	

Variable:	L2_weight									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	54.07	8.0%	14.98	2.0%	9.18	1.0%	10.13	1.0%	15.90	2.0%
2	44.29	2.0%	3.89	<1.0%	5.88	<1.0%	8.15	<1.0%	14.91	1.0%
3	39.90	2.0%	8.11	<1.0%	9.50	<1.0%	19.78	1.0%	21.14	1.0%
4	50.55	4.0%	11.39	1.0%	10.23	1.0%	23.89	2.0%	21.64	2.0%
5	47.03	2.0%	14.99	1.0%	15.48	1.0%	16.19	1.0%	25.43	1.0%
6	53.45	2.0%	20.72	1.0%	20.10	1.0%	25.57	1.0%	28.65	1.0%
7	56.61	3.0%	18.27	1.0%	17.04	1.0%	10.69	1.0%	22.19	1.0%
8	35.27	4.0%	13.84	2.0%	16.11	2.0%	13.62	2.0%	0.50	<1.0%
9	35.26	2.0%	13.85	1.0%	16.12	1.0%	18.13	1.0%	14.38	1.0%
10	56.96	6.0%	18.68	2.0%	17.47	2.0%	24.98	2.0%	5.95	1.0%
Average	47.34		13.87		13.71		17.11		17.07	
Std dev	8.35		5.06		4.62		6.42		8.69	

Variable:	L2_resin									
fold	Indicator		Median		Mean		Random		LOCF	
	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	7.63	11.0%	1.79	3.0%	1.56	2.0%	1.81	3.0%	0.72	1.0%
2	8.05	4.0%	2.23	1.0%	1.43	1.0%	1.77	1.0%	0.05	<1.0%
3	8.83	4.0%	3.16	1.0%	2.48	1.0%	2.46	1.0%	2.92	1.0%
4	8.43	7.0%	2.75	2.0%	2.51	2.0%	3.03	3.0%	3.23	3.0%
5	5.28	3.0%	1.89	1.0%	2.38	1.0%	2.33	1.0%	0.07	<1.0%
6	5.30	2.0%	1.88	1.0%	2.36	1.0%	1.16	1.0%	3.22	1.0%
7	6.50	4.0%	0.53	<1.0%	0.95	1.0%	1.61	1.0%	3.52	2.0%
8	6.52	8.0%	0.50	1.0%	0.91	1.0%	2.26	3.0%	2.23	3.0%
9	8.56	6.0%	2.86	2.0%	2.63	2.0%	2.61	2.0%	4.45	3.0%
10	9.74	1<1.0%	3.81	4.0%	3.19	3.0%	3.16	3.0%	2.06	2.0%
Average	7.48		2.14		2.04		2.22		2.25	
Std dev	1.52		1.07		0.77		0.64		1.52	

Variable	Thickness_swell_															
	MLR		PLS		NN		PCR		RT boosted		kNN		LASSO		Ridge	
fold	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	5.08	9.0%	2.28	4.0%	0.34	12.0%	2.38	4.0%	0.1	42.0%	4.31	19.0%	2.53	9.0%	2.31	87.0%
2	3.89	7.0%	1.71	3.0%	<0.00	1.0%	1.02	2.0%	<0.00	<1.0%	2.1	16.0%	2.56	9.0%	2.28	86.0%
3	2.27	4.0%	1.94	3.0%	0.65	48.0%	0.58	1.0%	0.09	7.0%	1.27	9.0%	2.57	9.0%	2.28	86.0%
4	2.22	4.0%	2.12	4.0%	0.12	53.0%	1.99	4.0%	<0.00	<1.0%	3.67	16.0%	2.51	9.0%	2.24	85.0%
5	6.33	11.0%	1.64	3.0%	<0.00	0.0%	0.63	1.0%	0.1	4.0%	3.96	17.0%	2.61	9.0%	2.33	88.0%
6	7.41	13.0%	1.89	3.0%	<0.00	1.0%	1.14	2.0%	<0.00	<1.0%	3.16	14.0%	2.71	10.0%	2.38	90.0%
7	2.24	4.0%	1.95	3.0%	<0.00	0.0%	2.37	4.0%	<0.00	<1.0%	3.91	17.0%	2.57	9.0%	2.3	87.0%
8	3.71	7.0%	2.66	5.0%	<0.00	1.0%	2.31	4.0%	<0.00	<1.0%	5.12	22.0%	2.55	9.0%	2.25	85.0%
9	4.98	9.0%	2.28	4.0%	0.07	31.0%	2.42	4.0%	0.1	42.0%	4.33	19.0%	2.53	9.0%	2.25	85.0%
10	4.6	8.0%	2.51	4.0%	0.16	6.0%	1.83	3.0%	<0.00	0.0%	3.75	16.0%	2.44	9.0%	2.23	85.0%
Average	4.27		2.1		0.13		1.67		0.04		3.56		2.56		2.29	
Std dev	1.77		0.33		0.21		0.75		0.05		1.13		0.07		0.05	

Variable	preasure															
	MLR		PLS		NN		PCR		RT boosted		kNN		LASSO		Ridge	
fold	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	304.62	54.0%	87.43	16.0%	58.13	47.0%	118.64	21.0%	<0.00	<1.0%	225.17	18.0%	124.46	8.0%	116.24	84.0%
2	234.03	41.0%	36.13	7.0%	50.02	108.0%	44.27	8.0%	<0.00	<1.0%	68.63	15.0%	130.94	9.0%	119.04	86.0%
3	114.52	20.0%	52.89	10.0%	1.18	2.0%	33.14	6.0%	<0.00	<1.0%	107.8	16.0%	126.33	9.0%	116.26	84.0%
4	110.21	19.0%	79.73	14.0%	31.3	26.0%	101.96	18.0%	<0.00	<1.0%	231.17	19.0%	121.04	8.0%	112.54	81.0%
5	362.63	65.0%	62.55	11.0%	36.53	30.0%	2.9	1.0%	<0.00	<1.0%	240.2	20.0%	124.34	8.0%	117.2	84.0%
6	433.49	77.0%	58.2	10.0%	6.24	5.0%	64.14	11.0%	<0.00	<1.0%	208.97	18.0%	130.86	9.0%	122.46	88.0%
7	115.86	21.0%	59.62	11.0%	15.12	13.0%	121.89	22.0%	<0.00	<1.0%	202.87	17.0%	127.48	9.0%	119.87	86.0%
8	194.47	35.0%	76.56	14.0%	33.25	27.0%	108.65	19.0%	<0.00	<1.0%	246.43	20.0%	129.27	9.0%	120.31	86.0%
9	315.32	56.0%	82.84	15.0%	18.37	15.0%	114.88	21.0%	2.02	17.0%	228.57	19.0%	128.52	9.0%	120.03	86.0%
10	297.85	53.0%	80.16	14.0%	32.12	26.0%	74.2	13.0%	<0.00	<1.0%	228.7	19.0%	122.5	8.0%	115.85	83.0%
Average	248.3		67.61		28.23		78.47		0.2		198.85		126.57		117.98	
Std dev	113.08		16.31		18.17		41.43		0.64		60.42		3.43		2.89	

Variable	QC_MOR															
	MLR		PLS		NN		PCR		RT boosted		kNN		LASSO		Ridge	
fold	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP	RMSEP	NRMSEP
1	35.62	6.0%	998.73	18.0%	976.28	17.0%	74.98	13.0%	<0.00	<1.0%	982.72	17.0%	36.76	6.0%	33.5	6.0%
2	25.83	5.0%	1280.5	22.0%	1315.6	23.0%	8.4	1.0%	6.97	12.0%	1359.2	23.0%	35.64	6.0%	31.88	5.0%
3	28.68	5.0%	1542.9	26.0%	1811.6	31.0%	131.81	23.0%	<0.00	<1.0%	1149.1	20.0%	37.38	6.0%	33.81	6.0%
4	32.17	6.0%	1752.4	32.0%	406.54	7.0%	151.55	27.0%	<0.00	<1.0%	1159.5	20.0%	40.8	7.0%	37.27	6.0%
5	43.05	8.0%	1670.4	30.0%	686.01	12.0%	109.42	20.0%	<0.00	<1.0%	1145.2	20.0%	40.81	7.0%	37.07	6.0%
6	67.21	12.0%	1671.5	30.0%	1340.1	23.0%	48.12	9.0%	1.54	3.0%	1149	20.0%	39.3	7.0%	35.52	6.0%
7	29.5	5.0%	1722.3	31.0%	1124.3	19.0%	105.06	19.0%	0.09	<1.0%	1517.2	26.0%	38.55	7.0%	35.36	6.0%
8	29.65	5.0%	1801.8	33.0%	503.46	9.0%	113.88	21.0%	<0.00	<1.0%	1702.6	29.0%	39.37	7.0%	35.54	6.0%
9	31.75	6.0%	1014.4	18.0%	992.19	17.0%	137.22	25.0%	0.09	<1.0%	973.62	17.0%	41	7.0%	36.92	6.0%
10	33.31	6.0%	1012.5	18.0%	967.71	17.0%	127.69	23.0%	0.17	<1.0%	980.3	17.0%	28.48	61.0%	28.67	61.0%
Average	35.68		1446.7		1012.4		100.81		0.89		1211.9		37.81		34.55	
Std dev	12.03		334.84		419.91		44.51		2.19		243.53		3.75		2.69	

Function of Every Simulated Variable

Variable Name	Cell name	Function
bin_temp	D4	=ROUND(20+0.1*NORM.INV(RAND(),0,0.32159),2)
Moisture_cont_bin1	D5	=ROUND(0.7+(0.01*1+(CHIQU.INV(RAND(),3)/7.0072-0.39)),2)
Moisture_cont_bin2	D6	=ROUND(0.7+0.01*(1+F.INV(RAND(),2,10)/3.7155-0.27),2)
bin_level1	D7	=ROUND(5+2*(T.INV(RAND(),2)/5.2422),2)
bin_level2	D8	=ROUND(5+2*(LOGNORM.INV(RAND(),0,0.5)/1.72-0.63),2)
dryer_heat	D9	=ROUND(200+2*NORM.INV(RAND(),0,0.32159),2)
dryer_speed	D10	=ROUND($J$3+0.5*NORM.INV(RAND(),0,0.32159),2)
dryer_rotation_Speed	D11	=ROUND(5+0.2*NORM.INV(RAND(),0,0.32159),2)
dryer_Airflow	D12	=ROUND(5+0.2*NORM.INV(RAND(),0,0.32159),2)
dyer_Weight	D13	=ROUND(($J$1*(1+(D5+D6)/2))+6*NORM.INV(RAND(),0,0.32159),2)
L1_weight	D14	=ROUND(((ROUNDDOWN(ROUNDDOWN(((D13/(1+((D5+D6)/2)))/10),0)/3,0)*10*2)*((D7/100)-0.05+1))+3*T.INV(RAND(),15)/3.2978,2)
L1_resin	D15	=ROUND(D14*0.1+0.1*NORM.INV(RAND(),0,0.32159),2)
L1_hardner	D16	=ROUND(D14*0.02+0.1*NORM.INV(RAND(),0,0.32159),2)
L1_melamine	D17	=ROUND(D14*0.06+0.1*NORM.INV(RAND(),0,0.32159),2)
L1_Moisture_cont	D18	=ROUND($J$4+0.01*NORM.INV(RAND(),0,0.32159),4)
L2_weight	D19	=ROUND((ROUNDUP(ROUNDUP(((D13/(1+((D5+D6)/2)))/10),0)/3,0)*10/2*2)*((D8/100)-0.05+1)+3*(F.INV(RAND(),15,5)/3.8494-0.35),2)
L2_resin	D20	=ROUND(D19*0.15*(1+0.03*NORM.INV(RAND(),0,0.32159)),2)
L2_hardner	D21	=ROUND(D19*0.01*(1+0.1*NORM.INV(RAND(),0,0.32159)),2)
L2_melamine	D22	=ROUND(D19*0.03*(1+0.05*NORM.INV(RAND(),0,0.32159)),2)

L2_Moisture_cont	D23	=ROUND($J$4+0.01*NORM.INV(RAND(),0,0.32159),4)
temp_outside	D24	Value going czclic between 10 and 25
Crew	D25	1 or 2
Product_no	D26	from 1 to 5
Planned_Thicknes	D27	20, 30 ,40 , 50 , 60
planned_Density	D28	between 500 and 1000
F1_lower_hight	D29	=ROUND(ROUNDDOWN(D27/3,0)*4+0.5*(CHIQU.INV(RAND(),10)/13.5556-0.72),2)
F1_mid_hight	D30	=ROUND(ROUNDUP(D27/3,0)/2*4+0.5*(CHIQU.INV(RAND(),10)/13.5556-0.72),2)+D29
F1_lower_weight	D31	=ROUND(AVERAGE((ROUNDDOWN(ROUNDDOWN(((D13/(1+((D5+D6)/2)))/10),0)/3,0)*10)+(D15+D16+D17)/2,((D14*$J$4/D18+D15+D16+D17)/2))+2*NORM.INV(RAND(),0,0.32159),2)
F1_mid_weight	D32	=ROUND((((ROUNDUP(ROUNDUP(((D13/(1+((D5+D6)/2)))/10),0)/3,0)*10)+(D20+D21+D22))/2)/$J$4*D23+D31+2*NORM.INV(RAND(),0,0.32159),2)
F1_width	D33	=ROUND(5100*(1+0.01*(LOGNORM.INV(RAND(),0,0.5)/1.72-0.63)),2)
F2_mid_hight	D34	=ROUND((ROUNDUP(D27/3,0)/2*4)*(1+(0.02*(CHIQU.INV(RAND(),10)/13.5556-0.72)))+D30,2)
F2_upper_hight	D35	=ROUND((ROUNDDOWN(D27/3,0)*4)*(1+(0.02*(CHIQU.INV(RAND(),10)/13.5556-0.72)))+D34,2)
F2_mid_weight	D36	=ROUND(((((ROUNDUP(ROUNDUP(((D13/(1+((D5+D6)/2)))/10),0)/3,0)*10)+D20+D21+D22)/2)/$J$4*D23+D32)*(1+0.02*(NORM.INV(RAND(),0,0.32159))),2)
F2_upper_weight	D37	=ROUND(((ROUNDDOWN(ROUNDDOWN(((D13/(1+((D5+D6)/2)))/10),0)/3,0)*10)/$J$4*D23+D36)*(1+0.02*(NORM.INV(RAND(),0,0.32159))),2)
F2_width	D38	=ROUND(5100*(1+0.01*(LOGNORM.INV(RAND(),0,0.5)/1.72-0.63)),2)
T1_1	D39	=ROUND(D41*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
T1_2	D40	=ROUND(D41*(1+0.01*(NORM.INV(RAND(),0,0.32159))),2)
T1_3	D41	=ROUND(D35*(1+0.07*(LOGNORM.INV(RAND(),0,0.5)/1.72-0.63)),2)

T1_4	D42	=ROUND(D41*(1+0.01*(NORM.INV(RAND(),0,0.32159))),2)
T1_5	D43	=ROUND(D41*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
W1_1	D44	=ROUND(D46*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
W1_2	D45	=ROUND(D46*(1+0.01*(NORM.INV(RAND(),0,0.32159))),2)
W1_3	D46	=ROUND(D37*(1+0.05*(CHIQU.INV(RAND(),3)/7.0059-0.39)),2)
W1_4	D47	=ROUND(D46*(1+0.01*(NORM.INV(RAND(),0,0.32159))),2)
W1_5	D48	=ROUND(D46*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
PP_Tickness_before	D49	=ROUND(D41*0.95*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
P_bar	D50	=ROUND(30+0.7*((1-GAMMA.INV(RAND(),0.5,0.3))/0.5293-1.67),2)
pp_speedref	D51	=ROUND($J$3*(1+0.05*(NORM.INV(RAND(),0,0.32159))),2)
pp_temp	D52	=ROUND(150*(1+0.05*(NORM.INV(RAND(),0,0.32159))),2)
PP_thickness_after	D53	=ROUND(D49*0.95*(1+0.05*(NORM.INV(RAND(),0,0.32159))),2)
P1_Tickness	D54	=ROUND(D53*0.7*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
P1_bar	D55	=ROUND(D50*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
P1_Centered	D56	=ROUND(0+(0.05*(F.INV(RAND(),10,10)/2.4414-0.47)),2)
P1_temp	D57	=ROUND(200*(1+0.05*(NORM.INV(RAND(),0,0.32159))),2)
P1_air_speed	D58	=ROUND(80*(1+0.1*(NORM.INV(RAND(),0,0.32159))),2)
P2_Tickness	D59	=ROUND(D54*0.7*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
P2_bar	D60	=ROUND(D50*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
P2_Centered	D61	=ROUND((0+D56)/2 +(0.05*(T.INV(RAND(),2)/5.2377)),2)
P2_temp	D62	=ROUND(D57*(1+0.05*(NORM.INV(RAND(),0,0.32159))),2)

P2_air_speed	D63	=ROUND(D58*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
P3_Tickness	D64	=ROUND(D59*0.7*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
P3_bar	D65	=ROUND(D50*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
P3_Centered	D66	=ROUND((0+D61)/2 +(0.05*(CHIQU.INV(RAND(),3)/7.0059-0.39)),2)
P3_temp	D67	=ROUND(D62*0.9*(1+0.05*(NORM.INV(RAND(),0,0.32159))),2)
P3_air_speed	D68	=ROUND(D58*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
P4_Tickness	D69	=ROUND(D64*0.8*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
P4_bar	D70	=ROUND(D50*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
P4_Centered	D71	=ROUND((0+D66)+(0.05*(NORM.INV(RAND(),0,0.32159))),2)
P4_temp	D72	=ROUND(D67*(1+0.05*(NORM.INV(RAND(),0,0.32159))),2)
P4_air_speed	D73	=ROUND(D58*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
T2_1	D74	=ROUND(D39/4*(D27+1)/D69*(1+0.03*D26*D25*(NORM.INV(RAND(),0,0.32159))),2)
T2_2	D75	=ROUND(D40/4*(D27+1)/D69*(1+0.02*(NORM.INV(RAND(),0,0.32159))),2)
T2_3	D76	=ROUND(D41/4*(1+D27)/D69*(1+0.01*(NORM.INV(RAND(),0,0.32159))),2)
T2_4	D77	=ROUND(D42/4*(1+D27)/D69*(1+0.02*(NORM.INV(RAND(),0,0.32159))),2)
T2_5	D78	=ROUND(D43/4*(1+D27)/D69*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
W2_1	D79	=ROUND((D44/$D$23*($J$4-$D$23*0.2))*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
W2_2	D80	=ROUND((D45/$D$23*($J$4-$D$23*0.2))*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
W2_3	D81	=ROUND((D46/$D$23*($J$4-$D$23*0.2))*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)

W2_4	D82	=ROUND((D47/$D$23*($J$4-$D$23*0.2))*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
W2_5	D83	=ROUND((D48/$D$23*($J$4-$D$23*0.2))*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
C_thickness	D84	=ROUND(AVERAGE(D74:D78)*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
C_bar	D85	=ROUND(D50*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
C_center	D86	=ROUND((0+D71)/2+(0.03*(NORM.INV(RAND(),0,0.32159))),2)
C_Temp	D87	=ROUND(100*(1+0.1*(NORM.INV(RAND(),0,0.32159))),2)
C_speed_air	D88	=ROUND(D73*(1+0.03*(NORM.INV(RAND(),0,0.32159))),2)
S_width1	D89	=ROUND((D38-5100+D86)*(1+0.01*(NORM.INV(RAND(),0,0.32159))),2)
S_width2	D90	=ROUND(((D38-100)/2+D86)*(1+0.01*(NORM.INV(RAND(),0,0.32159))),2)
S_width3	D91	=ROUND((D38-100+D86)*(1+0.01*(NORM.INV(RAND(),0,0.32159))),2)
S_lenght	D92	=ROUND(5005*(1+0.01*((1-GAMMA.INV(RAND(),0.5,0.3))/0.5293-1.67)),2)
S_thicknes	D93	=ROUND((D84-0.4)+0.05*(F.INV(RAND(),15,5)/3.8494-0.35),2)
E_width	D94	=ROUND(AVERAGE(D90-D89,D91-D90)*(1+0.01*D25*(NORM.INV(RAND(),0,0.32159))),2)
E_thicknes	D95	=ROUND(AVERAGE(D74,D75,D76,D77,D78)*(1+0.01*D25*(NORM.INV(RAND(),0,0.32159))),2)
E_lenght	D96	=ROUND(D92*(1+0.005*D25*(NORM.INV(RAND(),0,0.32159))),2)
E_weight	D97	=ROUND(AVERAGE(D79:D83)*(1+0.01*D25*(NORM.INV(RAND(),0,0.32159))),2)
E_moisture	D98	=ROUND(((D18+D23)/2)*D97/D37*(1+0.05*(NORM.INV(RAND(),0,0.32159))),4)
MOR	D99	Counter starting at 2000 per cell +1
MOR_DENS	D100	=ROUND(D115*(1+0.2*(NORM.INV(RAND(),0,0.32159))),2)

IB	D101	=ROUND(D114*2*(1+0.2*D25*(NORM.INV(RAND(),0,0.32159))),2)
IB_Dens	D102	=D101*50*$J$4
Surface	D103	=WURZEL(ROUND((D105+D106+D107+D108+D109)*(D113+D114+D115)*(1+0.05*(NORM.INV(RAND(),0,0.32159))),2)^2)
Thickness_swell	D114	=ROUND(ROUND((D15*D105+D20+(D31+D37)/(D32+D36)*D107+D57*D108+D68*D109+D80*D106+D87*D106+D88*D107+D25*D108)/5000+0.156,4)*(1+0.01*(NORM.INV(RAND(),0,0.32159))),4)*100
preasure	D115	=((D31+D37)/(D32+D36)*D105+D16*D106+D46*D106+D48*D106+D7*D107+D8*D108+D87*D105+D94+D95*D109+D96*D108+D97*D107+D98+D105*D109+D106*D106+D107*D109*D107+D108*D109)/100+450
MOR	D116	=ROUND(D20+D46+D60+D68+D70+D74+D76*0.5+D78+D85+D90+D98*0.5+D97+D95+D94*1.56+D83+D81,2)-1500

VBA Code for Data Simulation

```
Sub DataNULL()
Worksheets("sensordata").Activate
NoRows = Range("A1").Value
NULLRandom = Range("C2").Value
NULLBlock = Range("C3").Value
OUTLIERSRandom = Range("C4").Value
OUTLIERSBlock = Range("C5").Value

Worksheets("sensordata").Range("E8:CZ4483").ClearContents

Worksheets("Main_Sensor_Data").Range("E6:CZ4481").Copy _
  Worksheets("sensordata").Range("E8")

For i = 1 To 100
  Worksheets("sensordata").Activate
  Range("A8:CZ4481").Sort Key1:=Range("A8"), Order1:=xlAscending,
Header:=xlNo
  StartCell = Range("E8").Offset(0, i)
  EndCell = Range("E8").Offset(NULLRandom, i)
  Worksheets("sensordata").Range(Cells(8, 4 + i), Cells(8 + NULLRandom, 4 +
i)).Value = -100000.01
  ColumnNumber = i + 4
  ColumnLetter = Split(Cells(1, ColumnNumber).Address, "$")(1)

  Worksheets("sensordata").Range(Cells(9 + NULLRandom, 4 + i), Cells(9 +
NULLRandom + OUTLIERSRandom, 4 + i)).Formula _
    = "=RANDBETWEEN($" & ColumnLetter & "$2*100 ,$" & ColumnLetter &
"$3*100)/100"
  Worksheets("sensordata").Range(Cells(9 + NULLRandom, 4 + i), Cells(9 +
NULLRandom + OUTLIERSRandom, 4 + i)).Copy
  Worksheets("sensordata").Range(Cells(9 + NULLRandom, 4 + i), Cells(9 +
NULLRandom + OUTLIERSRandom, 4 + i)).PasteSpecial xlPasteValues
Next i
Range("A8:CZ4481").Sort Key1:=Range("B8"), Order1:=xlAscending,
Header:=xlNo
For i = 1 To 100
  MaxRange = NoRows - NULLBlock
  randomNumber = Int(MaxRange * Rnd) + 1
  Worksheets("sensordata").Range(Cells(8 + randomNumber, 4 + i), Cells(8 +
randomNumber + NULLRandom, 4 + i)).Value = -100000.01
```

```
  MaxRange2 = NoRows - OUTLIERSBlock
  randomNumber2 = Int(MaxRange2 * Rnd)
  If randomNumber2 - NULLBlock >= randomNumber Then
  If randomNumber2 + NULLBlock <= randomNumber + NULLBlock Then
  randomNumber2 = randomNumber2 + NULLBlock
  End If
  End If
  ColumnNumber = i + 4
  ColumnLetter = Split(Cells(1, ColumnNumber).Address, "$")(1)
  Worksheets("sensordata").Range(Cells(8 + randomNumber2, 4 + i), Cells(8 +
randomNumber2 + OUTLIERSBlock, 4 + i)).Formula _
    = "=RANDBETWEEN($" & ColumnLetter & "$2*100 ,$" & ColumnLetter &
"$3*100)/100"
  Worksheets("sensordata").Range(Cells(8 + randomNumber2, 4 + i), Cells(8 +
randomNumber2 + OUTLIERSBlock, 4 + i)).Copy
  Worksheets("sensordata").Range(Cells(8 + randomNumber2, 4 + i), Cells(8 +
randomNumber2 + OUTLIERSBlock, 4 + i)).PasteSpecial xlPasteValues
Next i

For i = 1 To 10
  If i = 1 Then
    randomNumber = Worksheets("qualitycontrol").Range("J2").Value - 1
  ElseIf i = 2 Then
    randomNumber = Worksheets("qualitycontrol").Range("J3").Value - 1
  Else
    firstValue = Worksheets("qualitycontrol").Range("J2").Value
    randomNumber = Int((NoRows - firstValue) * Rnd) + firstValue
  End If
  Worksheets("Main_Sensor_Data").Range("E6:CZ6").Offset(randomNumber,
0).Copy _
  Worksheets("sensordata").Range("E8").Offset(randomNumber, 0)

  For j = 1 To 100
    timelag = Range("D5").Offset(0, j).Value
    Worksheets("Main_Sensor_Data").Range("D6").Offset(randomNumber -
timelag, j).Copy _
    Worksheets("sensordata").Range("D8").Offset(randomNumber - timelag, j)

  Next j

Next i

MsgBox "Mischief managed"
End Sub
```

Program Code

index_reseach.py

```
import mysql.connector
import time
from DataFusion import DataFusionAlignmentMeasured
from DataFusion import DataFusionAlignmentPrediction
from QualityAssessment import QAMinMaxPrediction
from QualityAssessment import QAMinMaxMeasured
from QualityAssessment import QAGrubbsPrediction
from QualityAssessment import QAGrubbsMeasured
from DataImputationMethod import ImputationMethodSelection
from DataImputation import ImputationPrediction
from DataImputation import ImputationMeasured
from ModelMethod import ModelMethodSelection
from ModelMethodCalculation import ModelMethodCreation
from ModelImputation import ModelPrediction
from VariableSelection import Variablepreselection
from ResearchReport import research
import datetime


#  Connect to the Database
mydb = mysql.connector.connect(
          host ="localhost",
          user = "root",
          passwd = "Root",
          database = "program")

#  Object that communicates with the mySQL Database
mycursor = mydb.cursor()

# Variable declaration
secondsBetweenCheck = 5                         #Sets the time between new
checks (15min = 900sec)
LinespeedMedianValues = 1
PercentTrainingsData= 0.8
folds= 10
LookBack = 4000

Grubbsalpha = 0.01
NNAlpha = 0.05          # alpha for Neural network
HiddenLayers =(100,100,50)
```

```
PCANoComponents = 6
RTbDepth = 4
RtbEstimators = 300
kNNNeigbors = 3
LASSOAlpha = 0.05
RidgeAlpha = 0.05
LassoCutOff = 0.07

LastValueCheckOld = 0          # number of last input
LastValueCheckNew = 0

# Function looks up the last input
def LastInput():
    global LastValueCheckOld
    global LastValueCheckNew
    mycursor.execute("SELECT * FROM qualitycontrol WHERE inputNo =
(SELECT MAX(inputNo) FROM qualitycontrol)")
    myresult = mycursor.fetchall()
    # Store last index No
    for row in myresult:
        LastValueCheckNew = (int(row[1]))
    mydb.commit()
pass

#Function

def Program():
    sqlFormula1= "SELECT * FROM program.research"
    mycursor.execute(sqlFormula1)
    myresult = mycursor.fetchall()
    allDatabases = []
    for row in myresult:
        allDatabases.append(row)
    NoRuns = len(allDatabases)


    for RUN in range(1):

        sqlFormula21 = "DROP TABLE IF EXISTS `program`.`sensordata`"
        mycursor.execute(sqlFormula21)
        mydb.commit()
        sqlFormula3 = "CREATE TABLE `program`.`sensordata` SELECT *
FROM program.`{}`".format(str(allDatabases[RUN][1]))
        mycursor.execute(sqlFormula3)
        mydb.commit()
```

```
        sqlFormula41 = "DROP TABLE IF EXISTS `program`.`qualitycontrol`"
        mycursor.execute(sqlFormula41)
        mydb.commit()
        sqlFormula5 = "CREATE TABLE program.`qualitycontrol` SELECT *
FROM program.`{}`".format(str(allDatabases[RUN][2]))
        mycursor.execute(sqlFormula5)
        mydb.commit()

        sqlFormula61 = "DROP TABLE IF EXISTS
`program`.`fuseddataprediction`"
        mycursor.execute(sqlFormula61)
        mydb.commit()
        sqlFormula7 = "CREATE TABLE program.`fuseddataprediction` SELECT
* FROM program.`{}`".format(str(allDatabases[RUN][3]))
        mycursor.execute(sqlFormula7)
        mydb.commit()

        sqlFormula81 = "DROP TABLE IF EXISTS
`program`.`fuseddatameasured`"
        mycursor.execute(sqlFormula81)
        mydb.commit()
        sqlFormula9 = "CREATE TABLE program.`fuseddatameasured` SELECT
* FROM program.`{}`".format(str(allDatabases[RUN][4]))
        mycursor.execute(sqlFormula9)
        mydb.commit()

        sqlFormula101 = "DROP TABLE IF EXISTS `program`.`sensordistance`"
        mycursor.execute(sqlFormula101)
        mydb.commit()
        sqlFormula11 = "CREATE TABLE program.`sensordistance` SELECT *
FROM program.`{}`".format(str(allDatabases[RUN][5]))
        mycursor.execute(sqlFormula11)
        mydb.commit()

        number = int(allDatabases[RUN][0])
        name = str(allDatabases[RUN][7])
        CR = str(allDatabases[RUN][6])

    Method=[]
    Model=[]
    CalculatedModel=[]
    TimeBegin= datetime.datetime.now()
    global LastValueCheckOld
    global LastValueCheckNew
```

```
    global Grubbsalpha
    global LinespeedMedianValues
    global PercentTrainingsData
    global folds
    global LookBack
    global NNAlpha
    global HiddenLayers
    global PCANoComponents
    global RTbDepth
    global RtbEstimators
    global kNNNeigbors
    global LASSOAlpha
    global RidgeAlpha
    LastInput()
    if int(LastValueCheckNew) != int(LastValueCheckOld):      # Check if the new
index is new
        print("there is a new value ")
        LastValueCheckOld = LastValueCheckNew
        print("Check 1")
        DataFusionAlignmentMeasured(LinespeedMedianValues)            #
alignment and creation of new data.
        print("Check 2")
        DataFusionAlignmentPrediction(LinespeedMedianValues)
        print("Check 3")
        TimeAlignment= datetime.datetime.now()
        print("Alignment time:", TimeAlignment-TimeBegin)
        DeletedRowsM=QAMinMaxMeasured()                # Min Max Quality
assessment
        print("Check 4")
        DeletedRowsP=QAMinMaxPrediction()
        print("Check 5")
        TimeQAMinMax= datetime.datetime.now()
        print("QA Min Max time:",TimeQAMinMax-TimeAlignment)
        if CR == "C":
            OutlierRowsM=QAGrubbsMeasured(Grubbsalpha)              # Grubbs
outlier test
        else:
            OutlierRowsM = 0
        print("Check 6")
        if CR == "C":
            OutlierRowsP=QAGrubbsPrediction(Grubbsalpha)
        else:
            OutlierRowsP = 0
        print("Check 7")
        TimeGrubs= datetime.datetime.now()
```

```
        print("Grubbs time:",TimeGrubs-TimeQAMinMax)

Method=ImputationMethodSelection(PercentTrainingsData,folds,LookBack)
        print("Imputation Method =\n",Method)
        if CR == "C":
            ImputationMeasured(Method,LookBack)
        else:
            CR = CR
        print("Check 8")
        if CR == "C":
            ImputationPrediction(Method,LookBack)
        else:
            CR = CR
        print("Check 9")
        TimeImputation= datetime.datetime.now()
        print("Data Imputation time:",TimeImputation-TimeGrubs)
        SelectedVariables=Variablepreselection
(LookBack,LASSOAlpha,LassoCutOff)
        print("Check 9.5")

Model=ModelMethodSelection(PercentTrainingsData,folds,LookBack,NNAlpha
,HiddenLayers,PCANoComponents,RTbDepth,RtbEstimators,kNNNeigbors,LA
SSOAlpha,RidgeAlpha)
        print("Model =\n",Model)

CalculatedModel=ModelMethodCreation(SelectedVariables,Model,NNAlpha,Hi
ddenLayers,PCANoComponents,RTbDepth,RtbEstimators,kNNNeigbors,LASS
OAlpha,RidgeAlpha)
        print("Check 10")
        NoRowsPredicted =
ModelPrediction(SelectedVariables,CalculatedModel,Model)
        print("Check 11")
        TimePrediction= datetime.datetime.now()
        print("Updated: Imputation and Model: done= ",TimePrediction-
TimeImputation)
        #resach output start

research(number,name,TimeBegin,TimeAlignment,TimeQAMinMax,TimeGrubs
,TimeImputation,TimePrediction,Method,Model,DeletedRowsM,DeletedRowsP,
OutlierRowsM,OutlierRowsP,NoRowsPredicted)
        TimeEnd= datetime.datetime.now()
    else: # Start with the program
        print("no new value")
        DataFusionAlignmentPrediction(LinespeedMedianValues)
        QAMinMaxPrediction()
```

```
        QAGrubbsPrediction(Grubbsalpha)
        ImputationPrediction(Method,LookBack)
        TimeEnd= datetime.datetime.now()
    duration= TimeEnd-TimeBegin
    print("Time needed= {}".format(duration))
    print("Alignment time:", TimeAlignment-TimeBegin)
    print("QA Min Max time:",TimeQAMinMax-TimeAlignment)
    print("Grubbs time:",TimeGrubs-TimeQAMinMax)
    print("Data Imputation time:",TimeImputation-TimeGrubs)
    print("Updated: Imputation and Model: done= ",TimePrediction-
TimeImputation)
    time.sleep(secondsBetweenCheck)          # Secends before new DB
lookup
    Program()
pass

Program()
```

DataFusion.py

```
import mysql.connector
import datetime
import statistics

#  Connect ot the Database
mydb = mysql.connector.connect(
        host ="localhost",
        user = "root",
        passwd = "Root",
        database = "program")

#       Object that communicates with the mySQL Database
mycursor = mydb.cursor()

def DataFusionAlignmentMeasured(LinespeedMedianValues):
        numberOfValuesMedian = LinespeedMedianValues   # number of
intervals the programm fatches values
        #       Find all new Timestamps
        NewTime=[]
        sqlFormula1= "SELECT q.TimeStamp FROM qualitycontrol q Natural
Left Join fuseddatameasured f WHERE f.TimeStamp IS NULL ORDER BY
q.TimeStamp"
        mycursor.execute(sqlFormula1)
        myresult = mycursor.fetchall()
        for row in myresult:
```

```
                NewTime.append(row[0])
        mydb.commit()
        #       Find all Colums in the QC Table
        QCTableColums= []
        sqlFormula2= "SHOW COLUMNS FROM program.qualitycontrol"
        mycursor.execute(sqlFormula2)
        myresult = mycursor.fetchall()
        for row in myresult:
                QCTableColums.append(row[0])
        mydb.commit()
        #       Find all Colums in the Sensor Table (Production Data)
        PTableColums= []
        sqlFormula2= "SHOW COLUMNS FROM program.sensordata"
        mycursor.execute(sqlFormula2)
        myresult = mycursor.fetchall()
        for row in myresult:
                PTableColums.append(row[0])
        mydb.commit()
        #       Sensordistance
        SensorData=[]
        for i in range(len(PTableColums)-2): # -Timestamp and Linespeed =
number of Sensor
                sqlFormula4= "SELECT Distance FROM program.sensordistance
WHERE SensorName = '{}'".format(PTableColums[2+i])
                Sensor= PTableColums[2+i]
                mycursor.execute(sqlFormula4)
                myresult = mycursor.fetchall()
                for row in myresult:
                        SensorData.append(row[0])
                mydb.commit()
        #       Get the Linespeed to the time of QC Measurement
        Linespeed=[]
        for i in range(len(NewTime)):
                #Timestamp
                QCTime =NewTime[i-1]
                #Linespeed
                sqlFormula4 = "SELECT Linespeed, Stime FROM
program.sensordata WHERE STime <= '{}' ORDER BY STime DESC LIMIT
1".format(QCTime)
                mycursor.execute(sqlFormula4)
                myresult = mycursor.fetchall()
                for row in myresult:
                        Linespeed.append(float(row[0]))
                mydb.commit()
        #       Get the SensorData and QC Data with the timelag
```

```
        for i in range(len(NewTime)):
                SData = []
                QCdata=[]
                QCTime =NewTime[i-1]
                #       QC Data
                sqlFormula3 = "SELECT * FROM program.qualitycontrol WHERE
TimeStamp BETWEEN %s AND %s"
                mycursor.execute(sqlFormula3,(QCTime,QCTime))
                myresult = mycursor.fetchall()
                for row in myresult:
                        for j in range(len(row)-1):
                                QCdata.append(float(row[j+1]))
                mydb.commit()
                #       Sensordata
                Linespeed1=Linespeed[i]
                for k in range(len(PTableColums)-2):
                        SDataMedian = []
                        Time1 = QCTime -
datetime.timedelta(seconds=(SensorData[k]/Linespeed1))
                        sqlFormula5 = "SELECT {} FROM program.sensordata
WHERE STime <= '{}' AND '{}' IS NOT NULL ORDER BY STime DESC LIMIT
{}".format(PTableColums[2+k],Time1,PTableColums[2+k],numberOfValuesMed
ian)
                        mycursor.execute(sqlFormula5)
                        myresult = mycursor.fetchall()
                        for row in myresult:
                                for l in range(len(row)):
                                        #if time smaller than SData = NULL -100000
                                        try:
                                                test = float(row[l])
                                                SDataMedian.append(float(row[l]))
                                        except:
                                                test = row[l]
                        mydb.commit()
                        try:
                                SData.append(statistics.median(SDataMedian))
                        except:
                                print("Data Fusion 96", Time1, PTableColums[2+k])
                #       Create string with values
                DataString1 = ""
                for x in range(len(QCdata)-1):
                        DataString1 = DataString1 + ", " + "'" +
str(round(QCdata[x+1],2)) +"'"
                DataString1 = DataString1[1:(len(DataString1))]
                DataString2 = ""
```

```
                for x in range(len(SData)):
                        DataString2 = DataString2 + ", " +"'" +
str(round(SData[x],2))+"'"
                DataString2 = DataString2[1:(len(DataString2))]
                DataString= "'"+str(QCdata[0])+"'" + ", " + "'"+ str(Linespeed1)
+"'"+ ", " + "'0'" + ", " + DataString1 + ", " + DataString2
                #       insert vaule into new database
                sqlFormula6 ="INSERT INTO `program`.`fuseddatameasured`
VALUES ('{}', {})".format(QCTime,DataString)
                mycursor.execute(sqlFormula6)
                mydb.commit()
        mydb.commit()
pass
def DataFusionAlignmentPrediction(LinespeedMedianValues):
        numberOfValuesMedian = LinespeedMedianValues   # number of
intervals the programm fatches values
        #       Find all new Timestamps
        NewTime=[]
        sqlFormula1= "SELECT s.STime FROM program.sensordata s Left
outer Join program.fuseddataprediction f on f.TimeStamp = s.STime where
f.TimeStamp is null"
        mycursor.execute(sqlFormula1)
        myresult = mycursor.fetchall()
        for row in myresult:
                NewTime.append(row[0])
        mydb.commit()
        #       Find all Colums in the QC Table
        QCTableColums= []
        sqlFormula2= "SHOW COLUMNS FROM program.qualitycontrol"
        mycursor.execute(sqlFormula2)
        myresult = mycursor.fetchall()
        for row in myresult:
                QCTableColums.append(row[0])
        mydb.commit()
        #       Find all Colums in the Sensor Table (Production Data)
        PTableColums= []
        sqlFormula2= "SHOW COLUMNS FROM sensordata"
        mycursor.execute(sqlFormula2)
        myresult = mycursor.fetchall()
        for row in myresult:
                PTableColums.append(row[0])
        mydb.commit()
        #       Sensordistance
        SensorData=[]
```

```
        for i in range(len(PTableColums)-2): # -Timestamp and Linespeed =
number of Sensor
                sqlFormula4= "SELECT Distance FROM sensordistance WHERE
SensorName = '{}'".format(PTableColums[2+i])
                Sensor= PTableColums[2+i]
                mycursor.execute(sqlFormula4)
                myresult = mycursor.fetchall()
                for row in myresult:
                        SensorData.append(row[0])
                mydb.commit()
        #       Get the Linespeed to the time of QC Measurement
        Linespeed=[]
        for i in range(len(NewTime)):
                #Timestamp
                pTime =NewTime[i-1]
                #Linespeed
                sqlFormula4 = "SELECT Linespeed, STime FROM
program.sensordata WHERE STime <= '{}' ORDER BY STime DESC LIMIT
1".format(pTime)
                mycursor.execute(sqlFormula4)
                myresult = mycursor.fetchall()
                for row in myresult:
                        Linespeed.append(float(row[0]))
                mydb.commit()
        #       Get the SensorData with the timelag
        for i in range(len(NewTime)):
                SData = []
                pTime =NewTime[i-1]
                #       Sensordata
                Linespeed1=Linespeed[i]
                for k in range(len(PTableColums)-2):
                        Time1 = pTime -
datetime.timedelta(seconds=(SensorData[k]/Linespeed1))
                        sqlFormula5 = "SELECT {} FROM program.sensordata
WHERE STime <= '{}' ORDER BY STime DESC LIMIT
1".format(PTableColums[2+k],Time1)
                        mycursor.execute(sqlFormula5)
                        myresult = mycursor.fetchall()
                        if len(myresult) > 0:
                                try:
                                        float(myresult[0][0])
                                        for row in myresult:
                                                SData.append(round(float(row[0]),2))
                                except:
                                        SData.append("NULL")
```

```
                        else:
                                SData.append("NULL")
                        mydb.commit()
                #       insert vaule into new database
                DataString1 = ""
                for x in range(len(QCTableColums)-2):
                        DataString1 = DataString1 + ", " + "NULL"
                DataString1 = DataString1[1:(len(DataString1))]
                DataString2 = ""
                for x in range(len(SData)):
                        DataString2 = DataString2 + ", " +"" + str(SData[x])+""
                DataString2 = DataString2[1:(len(DataString2))]
                DataString= "'0"'+ ", " +"""+ str(Linespeed1) +"""+ ", " +
DataString1 + ", " + DataString2
                #print(DataString,"\n")
                sqlFormula6 ="INSERT INTO `program`.`fuseddataprediction`
VALUES ('{}', {})".format(pTime,DataString)
                mycursor.execute(sqlFormula6)
                mydb.commit()
        mydb.commit()
pass
```

QualityAssessment.py

```
import mysql.connector
import statistics
import datetime
import numpy
from outliers import smirnov_grubbs as grubbs
from scipy.stats import t
import math

#  Connect ot the Database
mydb = mysql.connector.connect(
        host ="localhost",
        user = "root",
        passwd = "Root",
        database = "program")

#       Object that communicates with the mySQL Database
mycursor = mydb.cursor()

def QAMinMaxPrediction():
        #       Find all Sensor Names in the Sensordistance table
        SensorNames= []
```

```
        sqlFormula1= "SHOW COLUMNS FROM program.sensordata"
        mycursor.execute(sqlFormula1)
        myresult = mycursor.fetchall()
        for row in range(len(myresult)-2):
                SensorNames.append(myresult[row+2][0])
        mydb.commit()
        #       Find the minimum value of the maximum Values
        maxSensors= []
        for i in range(len(SensorNames)):
                sqlFormula2 = "SELECT SensorMax, ProductMax, MachineMax
FROM program.sensordistance WHERE SensorName =
'{}'".format(SensorNames[i])
                mycursor.execute(sqlFormula2)
                myresult = mycursor.fetchall()
                for row in myresult:
                        maxSensors2=[]
                        for i in range(3):
                                try:
                                        maximum= float(row[i])
                                        maxSensors2.append(row[i])
                                except:
                                        test= 0
                maxSensors.append(min(maxSensors2))
        #       Find the maximum value of the minimum Values
        minSensors= []
        for i in range(len(SensorNames)):
                sqlFormula2 = "SELECT SensorMin, ProductMin, MachineMin
FROM program.sensordistance WHERE SensorName =
'{}'".format(SensorNames[i])
                mycursor.execute(sqlFormula2)
                myresult = mycursor.fetchall()
                for row in myresult:
                        minSensors2=[]
                        for i in range(3):
                                try:
                                        minimum= float(row[i])
                                        minSensors2.append(row[i])
                                except:
                                        test= 0
                minSensors.append(max(minSensors2))


        for i in range(len(SensorNames)):
                sqlFormula9= "UPDATE `program`.`fuseddataprediction` SET {}
= NULL WHERE Assessed= 0 AND {} < {} OR {} >
```

```
{}".format(SensorNames[i],SensorNames[i],minSensors[i],SensorNames[i],max
Sensors[i])
                mycursor.execute(sqlFormula9)
                mydb.commit()

        #       set assed value to 1
        sqlFormula10="UPDATE `program`.`fuseddataprediction` SET
`Assessed` = '1' WHERE Assessed= 0"
        mycursor.execute(sqlFormula10)
        mydb.commit()
        DeletedRows= 0
        return DeletedRows
pass
def QAMinMaxMeasured():
        #       Find all Sensor Names in the Sensordistance table
        SensorNames= []
        sqlFormula1= "SHOW COLUMNS FROM program.sensordata"
        mycursor.execute(sqlFormula1)
        myresult = mycursor.fetchall()
        for row in range(len(myresult)-2):
                row2= row + 2
                SensorNames.append(myresult[row2][0])
        mydb.commit()
        #       Find the minimum value of the maximum Values
        maxSensors= []
        for i in range(len(SensorNames)):
                sqlFormula2 = "SELECT SensorMax, ProductMax, MachineMax
FROM program.sensordistance WHERE SensorName =
'{}'".format(SensorNames[i])
                mycursor.execute(sqlFormula2)
                myresult = mycursor.fetchall()
                for row in myresult:
                        maxSensors2=[]
                        for i in range(3):
                                try:
                                        maximum= float(row[i])
                                        maxSensors2.append(row[i])
                                except:
                                        test= 0
                maxSensors.append(min(maxSensors2))
        #       Find the maximum value of the minimum Values
        minSensors= []
        for i in range(len(SensorNames)):
```

```
                sqlFormula2 = "SELECT SensorMin, ProductMin, MachineMin
FROM program.sensordistance WHERE SensorName =
'{}'".format(SensorNames[i])
                mycursor.execute(sqlFormula2)
                myresult = mycursor.fetchall()
                for row in myresult:
                        minSensors2=[]
                        for i in range(3):
                                try:
                                        minimum= float(row[i])
                                        minSensors2.append(row[i])
                                except:
                                        test= 0
                minSensors.append(max(minSensors2))

        for i in range(len(SensorNames)):
                sqlFormula9= "UPDATE `program`.`fuseddatameasured` SET {}
= NULL WHERE Assessed= 0 AND {} < {} OR {} >
{}".format(SensorNames[i],SensorNames[i],minSensors[i],SensorNames[i],max
Sensors[i])
                mycursor.execute(sqlFormula9)
                mydb.commit()

        #       set assed value to 1
        sqlFormula10="UPDATE `program`.`fuseddatameasured` SET
`Assessed` = '1' WHERE Assessed= 0"
        mycursor.execute(sqlFormula10)
        mydb.commit()
        DeletedRows= 0
        return DeletedRows
pass
def QAGrubbsPrediction(Grubbsalpha):
        #       Find all Sensor Names in the Sensordistance table
        SensorNames= []
        sqlFormula1= "SHOW COLUMNS FROM program.sensordata"
        mycursor.execute(sqlFormula1)
        myresult = mycursor.fetchall()
        for row in range(len(myresult)-2):
                row2= row + 2
                SensorNames.append(myresult[row2][0])
        mydb.commit()
        #       find the min and max Values of Grubbs outlier test
        ProductNumber= []
        sqlFormula111= "SELECT distinct Product_no FROM
program.fuseddataprediction WHERE Product_no is not NULL"
```

```
        mycursor.execute(sqlFormula111)
        myresult = mycursor.fetchall()
        for row in myresult:
                ProductNumber.append(row[0])
        mydb.commit()
        ProductNumber.append("NULL")

        for Prodtype in range(len(ProductNumber)):
                MinGrubbs=[]
                MaxGrubbs=[]
                for i in range(len(SensorNames)):
                        sqlFormula11="SELECT {} FROM
program.fuseddataprediction WHERE {} IS not NULL AND Product_no =
{}".format(SensorNames[i],SensorNames[i],ProductNumber[Prodtype])
                        mycursor.execute(sqlFormula11)
                        myresult = mycursor.fetchall()
                        AllValues=[]
                        for row in myresult:
                                AllValues.append(row[0])
                        mydb.commit()
                        try:
                                MeanValues = statistics.mean(AllValues)
                                StdevValues = statistics.stdev(AllValues)
                                if float(StdevValues) == 0:
                                        StdevValues = 1
                                alpha = Grubbsalpha
                                size = len(AllValues)
                                sigvalue=alpha/size
                                df = size-2
                                if df <= 0:
                                        df = asdf
                                tcrit = t.ppf(alpha, df)
                                gcrit = math.sqrt(((size-
1)*tcrit/math.sqrt(size*(df+tcrit**2)))**2)
                                MinGrubbs1 = float(MeanValues) - (float(gcrit) *
float(StdevValues))
                                MinGrubbs.append(float(MinGrubbs1))
                                MaxGrubbs1 = float(MeanValues) + (float(gcrit) *
float(StdevValues))
                                MaxGrubbs.append(float(MaxGrubbs1))
                        except:
                                MinGrubbs.append(-1000000)
                                MaxGrubbs.append(1000000)
                for i in range(len(SensorNames)):
```

```
                        sqlFormula12 = "UPDATE `program`.`fuseddataprediction`
SET {} = NULL WHERE Assessed= 1 And Product_no = {} AND {} < {} OR {} >
{}".format(SensorNames[i],ProductNumber[Prodtype],SensorNames[i],MinGrub
bs[i],SensorNames[i],MaxGrubbs[i])
                        mycursor.execute(sqlFormula12)
                        mydb.commit()

        sqlFormula13= "UPDATE `program`.`fuseddataprediction` SET
`Assessed` = '2' WHERE `Assessed` = '1'"
        mycursor.execute(sqlFormula13)
        mydb.commit()
        DeletedRows = 0
        return DeletedRows
pass
def QAGrubbsMeasured(Grubbsalpha):
        #       Find all Sensor Names in the Sensordistance table
        SensorNames= []
        sqlFormula1= "SHOW COLUMNS FROM program.sensordata"
        mycursor.execute(sqlFormula1)
        myresult = mycursor.fetchall()
        for row in range(len(myresult)-2):
                row2= row + 2
                SensorNames.append(myresult[row2][0])
        mydb.commit()
        #       find the min and max Values of Grubbs outlier test
        ProductNumber= []
        sqlFormula111= "SELECT distinct Product_no FROM
program.fuseddatameasured WHERE Product_no is not NULL"
        mycursor.execute(sqlFormula111)
        myresult = mycursor.fetchall()
        for row in myresult:
                ProductNumber.append(row[0])
        mydb.commit()

        for Prodtype in range(len(ProductNumber)):
                MinGrubbs=[]
                MaxGrubbs=[]
                for i in range(len(SensorNames)):
                        sqlFormula11="SELECT {} FROM
program.fuseddatameasured WHERE {} IS not NULL AND Product_no =
{}".format(SensorNames[i],SensorNames[i],ProductNumber[Prodtype])
                        mycursor.execute(sqlFormula11)
                        myresult = mycursor.fetchall()
                        AllValues=[]
                        for row in myresult:
```

```
                        AllValues.append(row[0])
                mydb.commit()
                try:
                        MeanValues = statistics.mean(AllValues)
                        StdevValues = statistics.stdev(AllValues)
                        if float(StdevValues) == 0:
                                StdevValues = 1
                        alpha = Grubbsalpha
                        size = len(AllValues)
                        sigvalue=alpha/size
                        df = size-2
                        if df <= 0:
                                df = asdf
                        tcrit = t.ppf(alpha, df)
                        gcrit = math.sqrt(((size-
1)*tcrit/math.sqrt(size*(df+tcrit**2)))**2)
                        MinGrubbs1 = float(MeanValues) - (float(gcrit) *
float(StdevValues))
                        MinGrubbs.append(float(MinGrubbs1))
                        MaxGrubbs1 = float(MeanValues) + (float(gcrit) *
float(StdevValues))
                        MaxGrubbs.append(float(MaxGrubbs1))
                except:
                        MinGrubbs.append(-1000000)
                        MaxGrubbs.append(1000000)

        for i in range(len(SensorNames)):
                sqlFormula12 = "UPDATE `program`.`fuseddatameasured`
SET {} = NULL WHERE Assessed= 1 And Product_no = {} AND {} < {} OR {} >
{}".format(SensorNames[i],ProductNumber[Prodtype],SensorNames[i],MinGrub
bs[i],SensorNames[i],MaxGrubbs[i])
                mycursor.execute(sqlFormula12)
                mydb.commit()
        sqlFormula13= "UPDATE `program`.`fuseddatameasured` SET
`Assessed` = '2' WHERE `Assessed` = '1'"
        mycursor.execute(sqlFormula13)
        mydb.commit()
        DeletedRows = 0
        return DeletedRows
pass
```

DataImputationMethod.py

```
import mysql.connector
import random
```

```
import math
from statistics import mean, median, stdev
import xlwt
import xlrd
from xlutils.copy import copy

#  Connect ot the Database
mydb = mysql.connector.connect(
        host ="localhost",
        user = "root",
        passwd = "Root",
        database = "program")

#       Object that communicates with the mySQL Database
mycursor = mydb.cursor()

def ImputationMethodSelection(PercentTrainingsData,folds,LookBack):
        #       define How much data is in the training data set
        trainingDataSet= PercentTrainingsData
        #       Find all Colums in the QC Table
        QCTableColums= []
        sqlFormula2= "SHOW COLUMNS FROM qualitycontrol"
        mycursor.execute(sqlFormula2)
        myresult = mycursor.fetchall()
        for row in myresult:
                QCTableColums.append(row[0])
        mydb.commit()
        #       Find all Colums in the Sensor Table (Production Data)
        PTableColums= []
        sqlFormula2= "SHOW COLUMNS FROM program.sensordata"
        mycursor.execute(sqlFormula2)
        myresult = mycursor.fetchall()
        for row in myresult:
                PTableColums.append(row[0])
        mydb.commit()
        #       Store Variables
        Variables=[]
        for i in range(len(QCTableColums)-2):
                Variables.append(QCTableColums[i+2])
        for i in range(len(PTableColums)-2):
                Variables.append(PTableColums[i+2])
        #       put Variables in a string
        #       Find all Colums in the fused data measutred Table
        FMTableColums= []
        sqlFormula3= "SHOW COLUMNS FROM program.fuseddatameasured"
```

```
        mycursor.execute(sqlFormula3)
        myresult = mycursor.fetchall()
        for row in myresult:
                FMTableColums.append(row[0])
        mydb.commit()

        CompleteDataSet=[]

        for i in range(len(FMTableColums)-4):
                sqlFormula4= "SELECT {} FROM program.fuseddatameasured
WHERE {} IS NOT NULL ORDER BY TimeStamp DESC Limit {}
".format(FMTableColums[i+4],FMTableColums[i+4],LookBack)
                mycursor.execute(sqlFormula4)
                myresult = mycursor.fetchall()
                Dataset= []
                for row in myresult:
                        Dataset.append(row[0])
                CompleteDataSet.append(Dataset)
                mydb.commit()

        CompletetrainingIndex=[]
        CompletevalidationIndex=[]

        for i in range(len(CompleteDataSet)):                           # for every
column
                NoRows = len(CompleteDataSet[i])
                if NoRows < 2:
                        print("DataImputation 72: NoRows smaller 2")
                elif int((NoRows*(1-trainingDataSet))) == 0:
                        NoValidationRows = 1
                        NoTrainingRows = NoRows-NoValidationRows
                else:
                        NoTrainingRows = NoRows - int((NoRows*(1-
trainingDataSet)))
                        NoValidationRows = int((NoRows*(1-trainingDataSet)))
                #       Create Index
                IndexRows=[]
                RandomIndex=[]
                for i in range(NoRows):
                        IndexRows.append(i)
                for i in range(NoRows):
                        RandomIndex.append(random.random())
                RandomizedIndex = [x for _,x in
sorted(zip(RandomIndex,IndexRows))]
```

```
                NoInK= int(NoRows/folds)

                trainingIndex=[]
                validationIndex=[]
                count=0

                # Create Validation index
                for j in range(folds):                                  # for
every fold
                        validationIndex1=[]
                        for k in range(NoValidationRows):               # for every
Row in the Validation set
                                if count+k >= NoRows:
                                        count=count-NoRows

        validationIndex1.append(RandomizedIndex[count+k])
                                else:

        validationIndex1.append(RandomizedIndex[count+k])
                        count= count + NoInK
                        validationIndex.append(validationIndex1)
                CompletevalidationIndex.append(validationIndex)
                #Create Training index
                count=0
                for j in range(folds):                                  # for
every fold
                        trainingIndex1=[]
                        for k in range(NoRows-NoValidationRows):        # for every
Row in the Training set
                                if count+NoValidationRows+k >= NoRows:
                                        count=count-NoRows

        trainingIndex1.append(RandomizedIndex[count+NoValidationRows+k])
                                else:

        trainingIndex1.append(RandomizedIndex[count+NoValidationRows+k])
                        count= count + NoInK
                        trainingIndex.append(trainingIndex1)
                CompletetrainingIndex.append(trainingIndex)

        # create report
        book = xlwt.Workbook(encoding="utf-8")
        sheet1 = book.add_sheet("Imputation")
        book.save("report.xls")
        # Claculate RMSEP for every imputation method
```

```
        Method=RMSEP(CompleteDataSet,CompletetrainingIndex,Completevali
dationIndex,folds)
        book= xlrd.open_workbook("report.xls")
        book1= copy(book)
        sheet1 = book1.get_sheet("Imputation")
        for i in range(len(Variables)):
                sheet1.write(i*(6+folds),0,"Variable:")
                sheet1.write(i*(6+folds),1,"{}".format(Variables[i]))
                sheet1.write(i*(6+folds)+(3+folds),0,"Average")
                sheet1.write(i*(6+folds)+(4+folds),0,"Std dev")
                for j in range(folds):
                        sheet1.write(i*(6+folds)+3+j,0,j+1)
        book1.save("report.xls")
        return Method
pass

def
RMSEPClaculation(CompleteDataSet,CompletetrainingIndex,Completevalidati
onIndex,ImputationValues,MethodNo,MethodName,folds):
        RMSEP=[]
        for i in range(len(CompletevalidationIndex)):
        # for every column
                NoRowsInColum=len(CompleteDataSet[i])
                NoTraining=len(CompletetrainingIndex[i])
                #NoValidation=len(CompletevalidationIndex[i][0])
                RMSEPindicator=[]

                RMSEP1=[]
                ObservedRangeAll=[]
                for j in range(len(CompletevalidationIndex[i])):   # folds 10
                        RMSEP2=[]
                        ObservedValues=[]
                        SumRMSEP2= 0
                        for k in range(len(CompletevalidationIndex[i][j])):
                                # for every Row in the Validation set

        Observed=float(CompleteDataSet[i][CompletevalidationIndex[i][j][k]])
                                        ObservedValues.append(Observed)
                                        Predicted= ImputationValues[i][j][k]
                                        RMSEP2.append((Observed-Predicted)**2)
                        for l in range(len(RMSEP2)):
                                        SumRMSEP2=SumRMSEP2+RMSEP2[l]
                        try:
                                        observedRange=round(max(ObservedValues)-
min(ObservedValues),2)
```

```
                        except:
                                observedRange= 0
                        ObservedRangeAll.append(observedRange)

        RMSEP1.append((round(math.sqrt(SumRMSEP2/len(RMSEP2)),2)))
                RMSEP.append(RMSEP1)

        book= xlrd.open_workbook("report.xls")
        book1= copy(book)
        sheet1 = book1.get_sheet("Imputation")
        for i in range(len(CompletevalidationIndex)):
                sheet1.write(1+i*(6+folds),0,"fold")
                sheet1.write(1+i*(6+folds),MethodNo,"{}".format(MethodName))
                sheet1.write(2+i*(6+folds),MethodNo,"RMSEP")
                sheet1.write(2+i*(6+folds),MethodNo+1,"NRMSEP")
                for j in range(len(RMSEP[i])):
                        if RMSEP[i][j] <0.01:
                                sheet1.write(j+3+(i*(6+folds)),MethodNo, "< 0.00")
                        else:
                                sheet1.write(j+3+(i*(6+folds)),MethodNo,
RMSEP[i][j])
                        try:
                                NRMSEP =
round(float(RMSEP[i][j])/float(ObservedRangeAll[j])*100,2)
                                if NRMSEP < 0.01:
                                        NRMSEP ="< 0.00"
                        except:
                                NRMSEP = "NA"
                        sheet1.write(j+3+(i*(6+folds)),MethodNo+1, NRMSEP)
                RMSEPindicator.append(round(mean(RMSEP[i]),2))
                StdDevRMSEP=round(stdev(RMSEP[i]),2)
                if RMSEPindicator[i] < 0.01:
                        sheet1.write((3+folds)+(i*(6+folds)),MethodNo, "< 0.00")
                else:
                        sheet1.write((3+folds)+(i*(6+folds)),MethodNo,
RMSEPindicator[i])
                if StdDevRMSEP <0.01:
                        sheet1.write((4+folds)+(i*(6+folds)),MethodNo, "< 0.00")
                else:
                        sheet1.write((4+folds)+(i*(6+folds)),MethodNo,
StdDevRMSEP)
        book1.save("report.xls")
        return RMSEPindicator
```

```
def
IndicatiorImputation(CompleteDataSet,CompletetrainingIndex,Completevalidati
onIndex,folds):
        ImputationValues=[]
        for i in range(len(CompletevalidationIndex)):
                        # no colums 103
                ImputationValues1=[]
                for j in range(len(CompletevalidationIndex[i])):
                # no folds 10
                        ImputationValues2=[]
                        for k in range(len(CompletevalidationIndex[i][j])):
                                # no validation varaibles
                                ImputationValues3= 0
                                ImputationValues2.append(ImputationValues3)
                        ImputationValues1.append(ImputationValues2)
                ImputationValues.append(ImputationValues1)

        MethodNo= 1
        MethodName= "Indicator"
        RMSEPindicator=RMSEPClaculation(CompleteDataSet,Completetrainin
gIndex,CompletevalidationIndex,ImputationValues,MethodNo,MethodName,fol
ds)
        return RMSEPindicator
pass

def
MedianImputation(CompleteDataSet,CompletetrainingIndex,Completevalidatio
nIndex,folds):
        Median0=[]
        for i in range(len(CompletetrainingIndex)):                # 8 varaibles
                Median1=[]
                for j in range(len(CompletetrainingIndex[i])):                        # 10
folds
                        Median2=[]
                        Median3=[]
                        for k in range(len(CompletetrainingIndex[i][j])):  # no in
each fold

        Median3.append(float(round(CompleteDataSet[i][CompletetrainingIndex
[i][j][k]],2)))
                        for k in range(len(CompletevalidationIndex[i][j])):
                                Median2.append(round(median(Median3),2))
                        Median1.append(Median2)
                Median0.append(Median1)
        ImputationValues=Median0
```

```
        MethodNo= 3
        MethodName= "Median"
        RMSEPindicator=RMSEPClaculation(CompleteDataSet,Completetrainin
gIndex,CompletevalidationIndex,ImputationValues,MethodNo,MethodName,fol
ds)
        return RMSEPindicator
pass
def
MeanImputation(CompleteDataSet,CompletetrainingIndex,CompletevalidationI
ndex,folds):
        Mean0=[]
        for i in range(len(CompletetrainingIndex)):                # 8 varaibles
                Mean1=[]
                for j in range(len(CompletetrainingIndex[i])):                  # 10
folds
                        Mean2=[]
                        Mean3=[]
                        for k in range(len(CompletetrainingIndex[i][j])):  # no in
each fold

        Mean3.append(float(round(CompleteDataSet[i][CompletetrainingIndex[i]
[j][k]],2)))
                        for k in range(len(CompletevalidationIndex[i][j])):
                                Mean2.append(round(mean(Mean3),2))
                        Mean1.append(Mean2)
                Mean0.append(Mean1)
        ImputationValues=Mean0
        MethodNo= 5
        MethodName= "Mean"
        RMSEPindicator=RMSEPClaculation(CompleteDataSet,Completetrainin
gIndex,CompletevalidationIndex,ImputationValues,MethodNo,MethodName,fol
ds)
        return RMSEPindicator
pass

def
RandomImputation(CompleteDataSet,CompletetrainingIndex,Completevalidati
onIndex,folds):
        Min0=[]
        Max0=[]
        for i in range(len(CompletetrainingIndex)):                # 8 varaibles
                Min1=[]
                Max1=[]
                for j in range(len(CompletetrainingIndex[i])):                  # 10
folds
```

```
                        Min2=[]
                        Max2=[]
                        for k in range(len(CompletetrainingIndex[i][j])):  # no in
each fold

        Min2.append(float(round(CompleteDataSet[i][CompletetrainingIndex[i][j][
k]],2)))

        Max2.append(float(round(CompleteDataSet[i][CompletetrainingIndex[i][j]
[k]],2)))
                        Min1.append(round(min(Min2),2))
                        Max1.append(round(max(Max2),2))
                Min0.append(Min1)
                Max0.append(Max1)
        ImputationValues= []
        for i in range(len(CompletevalidationIndex)):
                ImputationValues1= []
                for j in range(len(CompletevalidationIndex[i])):
                        ImputationValues2= []
                        for k in range(len(CompletevalidationIndex[i][j])):

        ImputationValues2.append(float(random.randint(int((Min0[i][j]-1)*100),
int((Max0[i][j])*100))/100))
                        ImputationValues1.append(ImputationValues2)
                ImputationValues.append(ImputationValues1)
        MethodNo= 7
        MethodName= "Random"
        RMSEPindicator=RMSEPClaculation(CompleteDataSet,Completetrainin
gIndex,CompletevalidationIndex,ImputationValues,MethodNo,MethodName,fol
ds)
        return RMSEPindicator
pass

def
LOCFImputation(CompleteDataSet,CompletetrainingIndex,CompletevalidationI
ndex,folds):
        sortedValidationIndex=[]
        for i in range(len(CompletevalidationIndex)):
                sortedValidationIndex1=[]
                for j in range(len(CompletevalidationIndex[i])):
                        toSort=CompletevalidationIndex[i][j]
                        sortedValidationIndex1.append(sorted(toSort))
                sortedValidationIndex.append(sortedValidationIndex1)
        ImputationValues=[]
```

```
        for i in range(len(CompleteDataSet)):                                   # for each
of 8 variables
                ImputationValues1=[]
                for j in range(len(CompletevalidationIndex[i])):   # folds 10
                        ImputationValues2=[]
                        for k in range(len(CompletevalidationIndex[i][j])):
        # no validation varaibles 2
                                ImputationValues3=
float(CompleteDataSet[i][sortedValidationIndex[i][j][k]-1])
                                ImputationValues2.append(ImputationValues3)
                        ImputationValues1.append(ImputationValues2)
                ImputationValues.append(ImputationValues1)
        MethodNo= 9
        MethodName= "LOCF"
        RMSEPindicator=RMSEPClaculation(CompleteDataSet,CompletetrainingIndex,CompletevalidationIndex,ImputationValues,MethodNo,MethodName,folds)
        return RMSEPindicator
pass

def
RMSEP(CompleteDataSet,CompletetrainingIndex,CompletevalidationIndex,folds):
        imputationMethod=["Indicator","Median","Mean","Random","LOCF"]
        RMSEP=[]
        x=IndicatiorImputation(CompleteDataSet,CompletetrainingIndex,CompletevalidationIndex,folds)
        RMSEP.append(x)
        x=MedianImputation(CompleteDataSet,CompletetrainingIndex,CompletevalidationIndex,folds)
        RMSEP.append(x)
        x=MeanImputation(CompleteDataSet,CompletetrainingIndex,CompletevalidationIndex,folds)
        RMSEP.append(x)
        x=RandomImputation(CompleteDataSet,CompletetrainingIndex,CompletevalidationIndex,folds)
        RMSEP.append(x)
        x=LOCFImputation(CompleteDataSet,CompletetrainingIndex,CompletevalidationIndex,folds)
        RMSEP.append(x)

        MethodSelection=[]
        for i in range(len(RMSEP[0])):              # 8 Varaibles
                MethodSelection1=[]
                for j in range(len(RMSEP)):                 # 7 Methods
```

```
                        MethodSelection1.append(float(RMSEP[j][i]))
                MethodSelection.append(min(MethodSelection1))

        count=0
        Method=[[]for i in range(len(RMSEP[0]))]
        for i in range(len(RMSEP[0])):                  # 8 Variables
                for j in range(len(RMSEP)):                     # 7 Methods
                        if RMSEP[j][i] == MethodSelection[i]:
                                Method[i]=imputationMethod[j]
                                count=+1

        return Method
pass
```

DataImputation.py

```
import mysql.connector
from statistics import median
import random

#  Connect ot the Database
mydb = mysql.connector.connect(
        host ="localhost",
        user = "root",
        passwd = "Root",
        database = "program")

#       Object that communicates with the mySQL Database
mycursor = mydb.cursor()

def ImputationMeasured(Method,LookBack):
        table="program.fuseddatameasured"
        #       Find all Colums in the QC Table
        QCTableColums= []
        sqlFormula1= "SHOW COLUMNS FROM qualitycontrol"
        mycursor.execute(sqlFormula1)
        myresult = mycursor.fetchall()
        for row in myresult:
                QCTableColums.append(row[0])
        mydb.commit()
        #       Find all Colums in the Sensor Table (Production Data)
        PTableColums= []
        sqlFormula2= "SHOW COLUMNS FROM sensordata"
        mycursor.execute(sqlFormula2)
        myresult = mycursor.fetchall()
```

```
        for row in myresult:
                PTableColums.append(row[0])
        mydb.commit()
        #       Store Variables
        Variables=[]
        for i in range(len(QCTableColums)-2):
                Variables.append(QCTableColums[i+2])
        for i in range(len(PTableColums)-2):
                Variables.append(PTableColums[i+2])

        # if this Mehtod than ...
                #
"Indicator","Median","Mean","Random","LOCF","LinearRegression","kNN1","Ma
ximumLikelyhood"
        for i in range(len(Method)):
                NullRows= []
                sqlFormula4="SELECT * FROM {} WHERE {} IS NULL ORDER
BY `TimeStamp`".format(table,Variables[i])
                mycursor.execute(sqlFormula4)
                myresult = mycursor.fetchall()
                for row in myresult:
                        NullRows.append(row)
                mydb.commit()

                if Method[i]== "Indicator":
                        IndicatorImputation(NullRows,table,Variables[i])
                elif Method[i]== "Median":
                        MedianImputation(NullRows,table,Variables[i],LookBack)
                elif Method[i]== "Mean":
                        MeanImputation(NullRows,table,Variables[i],LookBack)
                elif Method[i]== "Random":
                        RandomImputation(NullRows,table,Variables[i],LookBack)
                elif Method[i]== "LOCF":
                        LOCFImputation(NullRows,table,Variables[i])
                elif Method[i]== "LinearRegression":
                        LinearRegressionImputation(NullRows,table,Variables[i])
                elif Method[i]== "kNN1":
                        kNN1Imputation(NullRows,table,Variables[i])
                elif Method[i]== "MaximumLikelyhood":

        MaximumLikelyhoodImputation(NullRows,table,Variables[i])
                else:
                        print("Imputation = something is totally wrong!")
                for j in range(len(NullRows)):
```

```
                        sqlFormula= "UPDATE {} SET `Assessed` = '3' WHERE
(`TimeStamp` = '{}')".format(table,NullRows[j][0])
                        mycursor.execute(sqlFormula)
                        mydb.commit()
pass
def ImputationPrediction(Method,LookBack):
        table="program.fuseddataprediction"
        #       Find all Colums in the Sensor Table (Production Data)
        PTableColums= []
        sqlFormula2= "SHOW COLUMNS FROM sensordata"
        mycursor.execute(sqlFormula2)
        myresult = mycursor.fetchall()
        for row in myresult:
                PTableColums.append(row[0])
        mydb.commit()

        # if this Mehtod than ...
                #
"Indicator","Median","Mean","Random","LOCF","LinearRegression","kNN1"
        for i in range(len(PTableColums)-2):
                NullRows= []
                sqlFormula4="SELECT * FROM {} WHERE {} IS NULL ORDER
BY `TimeStamp`".format(table,PTableColums[i+2])
                mycursor.execute(sqlFormula4)
                myresult = mycursor.fetchall()
                for row in myresult:
                        NullRows.append(row)
                mydb.commit()

                if Method[i]== "Indicator":
                        IndicatorImputation(NullRows,table,PTableColums[i+2])
                elif Method[i]== "Median":

        MedianImputation(NullRows,table,PTableColums[i+2],LookBack)
                elif Method[i]== "Mean":

        MeanImputation(NullRows,table,PTableColums[i+2],LookBack)
                elif Method[i]== "Random":

        RandomImputation(NullRows,table,PTableColums[i+2],LookBack)
                elif Method[i]== "LOCF":
                        LOCFImputation(NullRows,table,PTableColums[i+2])
                elif Method[i]== "LinearRegression":

        LinearRegressionImputation(NullRows,table,PTableColums[i+2])
```

```
                elif Method[i]== "kNN1":
                        kNN1Imputation(NullRows,table,PTableColums[i+2])
                elif Method[i]== "MaximumLikelyhood":

        MaximumLikelyhoodImputation(NullRows,table,PTableColums[i+2])
                else:
                        print("Imputation = something is totally wrong!")
                for j in range(len(NullRows)):
                        sqlFormula= "UPDATE {} SET `Assessed` = '3' WHERE
(`TimeStamp` = '{}')".format(table,NullRows[j][0])
                        mycursor.execute(sqlFormula)
                        mydb.commit()
pass


def IndicatorImputation(NULLROWS,TABLE,VARIABLES):
        indicator=0
        for i in range(len(NULLROWS)):
                sqlFormula6="UPDATE {} SET {} = '{}' WHERE (`TimeStamp` =
'{}')".format(TABLE,VARIABLES,indicator,NULLROWS[i][0])
                mycursor.execute(sqlFormula6)
                mydb.commit()
pass
def MedianImputation(NULLROWS,TABLE,VARIABLES,LookBack):
        NoMedian =[0,0]
        MEDIAN= 0
        sqlFormula7="SELECT count({}) FROM {} WHERE {} IS NOT NULL
".format(VARIABLES,TABLE,VARIABLES)
        mycursor.execute(sqlFormula7)
        myresult = mycursor.fetchall()
        for row in myresult:
                if (row[0]%2)==0: # EVEN
                        NoMedian[0]=int(row[0]/2)
                        NoMedian[1]=int(row[0]/2)
                else:
                        NoMedian[0]=int(round((row[0]/2)-0.5,0))
                        NoMedian[1]=int(round((row[0]/2)+0.5,0))
        sqlFormula8="SELECT {} FROM {} WHERE {} IS NOT NULL ORDER
BY TimeStamp DESC Limit
{}".format(VARIABLES,TABLE,VARIABLES,LookBack)
        mycursor.execute(sqlFormula8)
        myresult = mycursor.fetchall()
        for i in range(len(NoMedian)):
                NoMedian[i]=myresult[NoMedian[i]][0]
        MEDIAN=median(NoMedian)
```

```
        for i in range(len(NULLROWS)):
                sqlFormula9="UPDATE {} SET {} = '{}' WHERE (`TimeStamp` =
'{}')".format(TABLE,VARIABLES,MEDIAN,NULLROWS[i][0])
                mycursor.execute(sqlFormula9)
                mydb.commit()
pass
def MeanImputation(NULLROWS,TABLE,VARIABLES,LookBack):
        MEAN=0
        sqlFormula10="SELECT avg({}) FROM
{}".format(VARIABLES,TABLE,LookBack)
        mycursor.execute(sqlFormula10)
        myresult = mycursor.fetchall()
        for row in myresult:
                MEAN=round(row[0],2)
        for i in range(len(NULLROWS)):
                sqlFormula11="UPDATE {} SET {} = '{}' WHERE `TimeStamp` =
'{}' ORDER BY TimeStamp DESC Limit
{}".format(TABLE,VARIABLES,MEAN,NULLROWS[i][0],LookBack)
                mycursor.execute(sqlFormula11)
                mydb.commit()
pass
def RandomImputation(NULLROWS,TABLE,VARIABLES,LookBack):
        Minimum=0
        Maximum=0
        sqlFormula12="SELECT max({}),min({}) FROM {} WHERE {} IS NOT
NULL ORDER BY TimeStamp DESC Limit
{}".format(VARIABLES,VARIABLES,TABLE,VARIABLES,LookBack)
        mycursor.execute(sqlFormula12)
        myresult = mycursor.fetchall()
        for row in myresult:
                Minimum=myresult[0][1]*100
                Maximum=myresult[0][0]*100
        for i in range(len(NULLROWS)):
                RandomVariable=random.randint(Minimum,Maximum)/100
                sqlFormula13="UPDATE {} SET {} = '{}' WHERE (`TimeStamp` =
'{}')".format(TABLE,VARIABLES,RandomVariable,NULLROWS[i][0])
                mycursor.execute(sqlFormula13)
                mydb.commit()
pass
def LOCFImputation(NULLROWS,TABLE,VARIABLES):
        for i in range(len(NULLROWS)):
                LastObservation= 0
                Time=NULLROWS[i][0]
                sqlFormula14="SELECT {} FROM {} WHERE TimeStamp < '{}'
ORDER BY TimeStamp DESC Limit 1".format(VARIABLES,TABLE,Time)
```

```
                mycursor.execute(sqlFormula14)
                myresult = mycursor.fetchall()
                for row in myresult:
                        LastObservation= row[0]
                sqlFormula15="UPDATE {} SET {} = '{}' WHERE (`TimeStamp` =
'{}')".format(TABLE,VARIABLES,LastObservation,NULLROWS[i][0])
                mycursor.execute(sqlFormula15)
                mydb.commit()
pass
def LinearRegressionImputation(NULLROWS,TABLE,VARIABLES):
        x=0
pass
def kNN1Imputation(NULLROWS,TABLE,VARIABLES):
        x=0
pass
def MaximumLikelyhoodImputation(NULLROWS,TABLE,VARIABLES):
        x=0
pass
```

VariableSelection.py

```
import mysql.connector
from sklearn.linear_model import Lasso
import math

#  Connect ot the Database
mydb = mysql.connector.connect(
        host ="localhost",
        user = "root",
        passwd = "Root",
        database = "program")
#       Object that communicates with the mySQL Database
mycursor = mydb.cursor()


#LookBack= 4000
#LASSOAlpha = 0.01
#LassoCutOff= 1


def Variablepreselection (LookBack,LASSOAlpha, LassoCutOff):
        #       Find all Colums in the QC Table
        QCTableColums= []
        sqlFormula2= "SHOW COLUMNS FROM qualitycontrol"
        mycursor.execute(sqlFormula2)
```

```
        myresult = mycursor.fetchall()
        for row in myresult:
                QCTableColums.append(row[0])
        mydb.commit()
        #       Find all Colums in the Sensor Table (Production Data)
        PTableColums= []
        sqlFormula2= "SHOW COLUMNS FROM sensordata"
        mycursor.execute(sqlFormula2)
        myresult = mycursor.fetchall()
        for row in myresult:
                PTableColums.append(row[0])
        mydb.commit()
        #       Store Variables
        Variables=[]
        for i in range(len(QCTableColums)-2):
                Variables.append(QCTableColums[i+2])
        for i in range(len(PTableColums)-2):
                Variables.append(PTableColums[i+2])
        #       put Variables in a string
        VariableString = ""
        for x in range(len(Variables)-1):
          VariableString = VariableString +"`"+ Variables[x] +"`"+ " IS NOT NULL
AND "
        VariableString = VariableString+"`" + Variables[len(Variables)-1]+"`"
        #       Select all rows with no NULLS
        sqlFormula1= "SELECT * FROM program.fuseddatameasured WHERE
{} IS NOT NULL ORDER BY TimeStamp DESC Limit {} ".format(VariableString,
LookBack)
        mycursor.execute(sqlFormula1)
        myresult = mycursor.fetchall()
        CompleteDataSet=[]
        for row in myresult:
                Dataset= []
                for i in range(len(row)):
                        Dataset.append(row[i])
                CompleteDataSet.append(Dataset)
        mydb.commit()

        X=[]
        for i in range(len(CompleteDataSet)):
                Xi=[]
                Xi.append(float(CompleteDataSet[i][2])) # Linespeed
                for j in range (len(PTableColums)-2):
                        variablePlace= 4 + len(QCTableColums)-2
                        Xi.append(float(CompleteDataSet[i][variablePlace+j]))
```

```
                X.append(Xi)

        SelectedVariables = []
        for i in range(len(QCTableColums)-2):
                Y=[]
                for k in range(len(CompleteDataSet)):
                        Y.append(float(CompleteDataSet[k][4+i]))
                lasso = Lasso(alpha=LASSOAlpha)
                lasso.fit(X,Y)
                SelectedVariables1=[]
                for j in range(len(Variables)-(len(QCTableColums)-2)):
                        if math.sqrt(lasso.coef_[j]**2) >= LassoCutOff:

        SelectedVariables1.append(Variables[j+(len(QCTableColums)-2)])
                SelectedVariables.append(SelectedVariables1)

        #print(len(SelectedVariables[0]),len(SelectedVariables[1]),len(SelectedV
ariables[2]))
        return SelectedVariables


#SelectedVariables=Variablepreselection
(LookBack,LASSOAlpha,LassoCutOff)
```

ModelMethod.py

```
import mysql.connector
import random
import math
from statistics import mean, stdev
from sklearn.linear_model import LinearRegression
from sklearn.cross_decomposition import PLSRegression
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.decomposition import PCA
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.neighbors import KNeighborsClassifier,KNeighborsRegressor
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
import xlwt
import xlrd
from xlutils.copy import copy
```

```
#  Connect ot the Database
mydb = mysql.connector.connect(
        host ="localhost",
        user = "root",
        passwd = "Root",
        database = "program")

#       Object that communicates with the mySQL Database
mycursor = mydb.cursor()

def
ModelMethodSelection(PercentTrainingsData,folds,LookBack,NNAlpha,Hidden
Layers,PCANoComponents,RTbDepth,RtbEstimators,kNNNeigbors,LASSOAlp
ha,RidgeAlpha):
        #       define How much data is in the training data set
        trainingDataSet= PercentTrainingsData
        #       Find all Colums in the QC Table
        QCTableColums= []
        sqlFormula2= "SHOW COLUMNS FROM qualitycontrol"
        mycursor.execute(sqlFormula2)
        myresult = mycursor.fetchall()
        for row in myresult:
                QCTableColums.append(row[0])
        mydb.commit()
        #       Find all Colums in the Sensor Table (Production Data)
        PTableColums= []
        sqlFormula2= "SHOW COLUMNS FROM sensordata"
        mycursor.execute(sqlFormula2)
        myresult = mycursor.fetchall()
        for row in myresult:
                PTableColums.append(row[0])
        mydb.commit()
        #       Store Variables
        Variables=[]
        for i in range(len(QCTableColums)-2):
                Variables.append(QCTableColums[i+2])
        for i in range(len(PTableColums)-2):
                Variables.append(PTableColums[i+2])
        #       put Variables in a string
        VariableString = ""
        for x in range(len(Variables)-1):
          VariableString = VariableString +"`"+ Variables[x] +"`"+ " IS NOT NULL
AND "
        VariableString = VariableString+"`" + Variables[len(Variables)-1]+"`"
```

```
        #       Select all rows with no NULLS
        sqlFormula1= "SELECT * FROM program.fuseddatameasured WHERE
{} IS NOT NULL ORDER BY TimeStamp DESC Limit {} ".format(VariableString,
LookBack)
        mycursor.execute(sqlFormula1)
        myresult = mycursor.fetchall()
        CompleteDataSet=[]
        for row in myresult:
                Dataset= []
                for i in range(len(row)):
                        Dataset.append(row[i])
                CompleteDataSet.append(Dataset)
        mydb.commit()
        #       Count of rows
        NoRows= len(CompleteDataSet)
        if NoRows < 2:
                print("DataImputation 74: NoRows smaller 2")
        elif int((NoRows*(1-trainingDataSet))) == 0:
                NoValidationRows = 1
                NoTrainingRows = NoRows-NoValidationRows
        else:
                NoTrainingRows = NoRows - int((NoRows*(1-trainingDataSet)))
                NoValidationRows = int((NoRows*(1-trainingDataSet)))
        #       Create Index
        IndexRows=[]
        RandomIndex=[]
        for i in range(NoRows):
                IndexRows.append(i)
        for i in range(NoRows):
                RandomIndex.append(random.random())
        RandomizedIndex = [x for _,x in sorted(zip(RandomIndex,IndexRows))]
        #       trainingSet
        NoInK= int(NoRows/folds)
        #factor=int(NoValidationRows/NoInK)
        trainingIndex=[]
        validationIndex=[]
        count=0
        for i in range(folds):
                validationIndex1=[]
                for j in range(NoValidationRows):
                        if count+j >= NoRows:
                                count=count-NoRows

        validationIndex1.append(RandomizedIndex[count+j])
                        else:
```

```
        validationIndex1.append(RandomizedIndex[count+j])
                count= count + NoInK
                validationIndex.append(validationIndex1)
        count=0
        for i in range(folds):
                trainingIndex1=[]
                for j in range(NoRows-NoValidationRows):
                        if count+NoValidationRows+j >= NoRows:
                                count=count-NoRows

        trainingIndex1.append(RandomizedIndex[count+NoValidationRows+j])
                        else:

        trainingIndex1.append(RandomizedIndex[count+NoValidationRows+j])
                count= count + NoInK
                trainingIndex.append(trainingIndex1)
        #Sort index
        sortedValidationIndex=[]
        for i in range(len(validationIndex)):
                toSort=validationIndex[i]
                sortedValidationIndex.append(sorted(toSort))
        sortedTrainingIndex=[]
        validationIndex=sortedTrainingIndex
        for i in range(len(trainingIndex)):
                toSort=trainingIndex[i]
                sortedTrainingIndex.append(sorted(toSort))
        trainingIndex=sortedTrainingIndex
        #Claculate RMSEP for every imputation method
        book= xlrd.open_workbook("report.xls")
        book1= copy(book)
        sheet1 = book1.add_sheet("Model")
        sheet1 = book1.get_sheet("Model")
        for i in range(len(QCTableColums)-2):
                sheet1.write(i*(6+folds),0,"Variable")
                sheet1.write(i*(6+folds)+1,0,"fold")
                sheet1.write(i*(6+folds),1,"{}".format(QCTableColums[i+2]))
                sheet1.write(i*(6+folds)+(3+folds),0,"Average")
                sheet1.write(i*(6+folds)+(4+folds),0,"Std dev")
                for j in range(folds):
                        sheet1.write(i*(6+folds)+3+j,0,j+1)
        book1.save("report.xls")
        Model=RMSEP(QCTableColums,PTableColums,CompleteDataSet,trainingIndex,validationIndex,folds,NNAlpha,HiddenLayers,PCANoComponents,RTbDepth,RtbEstimators,kNNNeigbors,LASSOAlpha,RidgeAlpha)
```

```
        return Model
pass
def
XYCreation(QCTableColums,PTableColums,CompleteDataSet,trainingIndex,v
alidationIndex):
        X=[]
        XValidation=[]
        Y=[]
        YRange1=[]
        for j in range(len(trainingIndex)):
                Xi=[]
                Xi.append(CompleteDataSet[trainingIndex[j]][2])         #
Linespeed
                for k in range(len(PTableColums)-2):
                        variablePlace= 4 + len(QCTableColums)-2

        Xi.append(CompleteDataSet[trainingIndex[j]][variablePlace+k])
                X.append(Xi)
        for j in range(len(trainingIndex)):
                Yi=[]
                for k in range(len(QCTableColums)-2):
                        variablePlace= 4

        Yi.append(float(CompleteDataSet[trainingIndex[j]][variablePlace+k]))
                Y.append(Yi)
                YRange1.append(max(Yi)-min(Yi))
        for j in range(len(validationIndex)):
                XiValidation=[]
                XiValidation.append(CompleteDataSet[validationIndex[j]][2]) #
Linespeed
                for k in range(len(PTableColums)-2):
                        variablePlace= 4 + len(QCTableColums)-2

        XiValidation.append(float(CompleteDataSet[validationIndex[j]][variablePl
ace+k]))
                XValidation.append(XiValidation)
        Output=[]
        Output.append(X)
        Output.append(XValidation)
        Output.append(Y)
        Output.append(YRange1)
        return Output
pass
```

```
def
RMSEPReport(QCTableColums,PTableColums,CompleteDataSet,trainingInde
x,validationIndex,ModelNumber,ModelName,Y_pred,i,YRange1,folds):
        RMSEP1=[]
        for j in range(len(QCTableColums)-2):
                RMSEP2=[]
                for k in range(len(validationIndex)):
                        RMSEP3=math.sqrt((Y_pred[k][j]-
float(CompleteDataSet[validationIndex[k]][4+j]))**2)
                        RMSEP2.append(RMSEP3)
                RMSEP1.append(mean(RMSEP2))
        # Report
        book= xlrd.open_workbook("report.xls")
        book1= copy(book)
        sheet1 = book1.get_sheet("Model")
        for j in range(len(RMSEP1)):

        sheet1.write(1+j*(6+folds),ModelNumber,"{}".format(ModelName))
                sheet1.write(2+j*(6+folds),ModelNumber,"RMSEP")
                sheet1.write(2+j*(6+folds),ModelNumber+1,"NRMSEP")
                sheet1.write(3+j*(6+folds)+i,ModelNumber,round(RMSEP1[j],2))
                try:
                        NRMSEP2= round(RMSEP1[j]/YRange1[j]*100,2)
                except:
                        NRMSEP2="NA"
                sheet1.write(3+j*(6+folds)+i,ModelNumber+1,NRMSEP2)
        book1.save("report.xls")
        return RMSEP1
pass
def
MLR(QCTableColums,PTableColums,CompleteDataSet,trainingIndex,validatio
nIndex,folds):
        RMSEP=[]
        ModelNumber=1
        ModelName= "MLR"
        for i in range(len(trainingIndex)):
                # Create X and Y values

        Output=XYCreation(QCTableColums,PTableColums,CompleteDataSet,t
rainingIndex[i],validationIndex[i])
                X=Output[0]
                XValidation=Output[1]
                Y=Output[2]
                YRange1=Output[3]
                # predict Y Values validation
```

```
                mlr1=LinearRegression().fit(X, Y)
                Y_pred=mlr1.predict(XValidation)
                #calculate RMSEP

        RMSEP1=RMSEPReport(QCTableColums,PTableColums,CompleteDat
aSet,trainingIndex[i],validationIndex[i],ModelNumber,ModelName,Y_pred,i,YRa
nge1,folds)
                RMSEP.append(RMSEP1)
        RMSEPpred=[]
        StdDevRMSEP=[]
        for i in range(len(RMSEP[0])):
                RMSEPpred1=[]
                for j in range(len(RMSEP)):
                        RMSEPpred2=RMSEP[j][i]
                        RMSEPpred1.append(RMSEPpred2)
                RMSEPpred.append(mean(RMSEPpred1))
                StdDevRMSEP1=round(stdev(RMSEPpred1),2)
                StdDevRMSEP.append(StdDevRMSEP1)
        book= xlrd.open_workbook("report.xls")
        book1= copy(book)
        sheet1 = book1.get_sheet("Model")
        for i in range(len(RMSEPpred)):
                if round(RMSEPpred[i]) < 0.01:

        sheet1.write((3+folds)+i*(6+folds),ModelNumber,round(RMSEPpred[i],2)
)
                else:
                        sheet1.write((3+folds)+i*(6+folds),ModelNumber, "<0.00")

        sheet1.write((4+folds)+i*(6+folds),ModelNumber,StdDevRMSEP[i])
        book1.save("report.xls")
        return RMSEPpred
pass
def
PLS(QCTableColums,PTableColums,CompleteDataSet,trainingIndex,validatio
nIndex,folds):
        RMSEP=[]
        ModelNumber=3
        ModelName= "PLS"
        for i in range(len(trainingIndex)):

        Output=XYCreation(QCTableColums,PTableColums,CompleteDataSet,t
rainingIndex[i],validationIndex[i])
                X=Output[0]
                XValidation=Output[1]
```

```
                Y=Output[2]
                YRange1=Output[3]
                pls1 = PLSRegression(copy=True, max_iter=500, scale=True,
tol=1e-06)
                pls1.fit(X, Y)
                Y_pred=pls1.predict(XValidation)

        RMSEP1=RMSEPReport(QCTableColums,PTableColums,CompleteDat
aSet,trainingIndex[i],validationIndex[i],ModelNumber,ModelName,Y_pred,i,YRa
nge1,folds)
                RMSEP.append(RMSEP1)
        RMSEPpred=[]
        StdDevRMSEP=[]
        for i in range(len(RMSEP[0])):
                RMSEPpred1=[]
                for j in range(len(RMSEP)):
                        RMSEPpred2=RMSEP[j][i]
                        RMSEPpred1.append(RMSEPpred2)
                RMSEPpred.append(mean(RMSEPpred1))
                StdDevRMSEP1=round(stdev(RMSEPpred1),2)
                StdDevRMSEP.append(StdDevRMSEP1)
        book= xlrd.open_workbook("report.xls")
        book1= copy(book)
        sheet1 = book1.get_sheet("Model")
        for i in range(len(RMSEPpred)):
                if round(RMSEPpred[i]) > 0.01:

        sheet1.write((3+folds)+i*(6+folds),ModelNumber,round(RMSEPpred[i],2)
)
                else:
                        sheet1.write((3+folds)+i*(6+folds),ModelNumber, "<0.00")

        sheet1.write((4+folds)+i*(6+folds),ModelNumber,StdDevRMSEP[i])
        book1.save("report.xls")
        return RMSEPpred
pass
def
NN(QCTableColums,PTableColums,CompleteDataSet,trainingIndex,validationI
ndex,folds,NNAlpha,HiddenLayers):
        RMSEP=[]
        ModelNumber=5
        ModelName="NN"
        for i in range(len(trainingIndex)):
                X=[]
                XValidation=[]
```

```
                Y=[]
                for j in range(len(trainingIndex[i])):
                        Xi=[]
                        Xi.append(float((CompleteDataSet[trainingIndex[i][j]][2])))
        # Linespeed
                        for k in range(len(PTableColums)-2):
                                variablePlace= 4 + len(QCTableColums)-2

        Xi.append(float(CompleteDataSet[trainingIndex[i][j]][variablePlace+k]))
                        X.append(Xi)
                for j in range(len(validationIndex[i])):
                        XiValidation=[]

        XiValidation.append(CompleteDataSet[validationIndex[i][j]][2])        #
Linespeed
                        for k in range(len(PTableColums)-2):
                                variablePlace= 4 + len(QCTableColums)-2

        XiValidation.append(float(CompleteDataSet[validationIndex[i][j]][variable
Place+k]))
                        XValidation.append(XiValidation)
                Y_pred=[]
                YRange=[]
                for j in range(len(QCTableColums)-2):
                        Y=[]
                        for k in range(len(trainingIndex[i])):

        Y.append(float(CompleteDataSet[trainingIndex[i][k]][4+j]))
                        YRange.append(max(Y)-min(Y))
                        nn = MLPRegressor(solver='lbfgs',
alpha=NNAlpha,hidden_layer_sizes=HiddenLayers, random_state=1)
                        nn.fit(X, Y)
                        Y_pred1= nn.predict(XValidation)
                        Y_pred.append(Y_pred1)
                RMSEP1=[]
                for j in range(len(QCTableColums)-2):
                        RMSEP2=[]
                        for k in range(len(validationIndex[i])):
                                RMSEP3=math.sqrt((Y_pred[j][k]-
float(CompleteDataSet[validationIndex[i][k]][4+j]))**2)
                                RMSEP2.append(RMSEP3)
                        RMSEP1.append(mean(RMSEP2))
                book= xlrd.open_workbook("report.xls")
                book1= copy(book)
                sheet1 = book1.get_sheet("Model")
```

```
                for j in range(len(RMSEP1)):
                        sheet1.write(1+j*(6+folds),ModelNumber,ModelName)
                        sheet1.write(2+j*(6+folds),ModelNumber,"RMSEP")
                        sheet1.write(2+j*(6+folds),ModelNumber+1,"NRMSEP")

        sheet1.write(3+j*(6+folds)+i,ModelNumber,round(RMSEP1[j],2))
                        try:
                                NRMSEP=round(RMSEP1[j]/YRange[j] *100,2)
                        except:
                                NRMSEP = "NA"
                        sheet1.write(3+j*(6+folds)+i,ModelNumber+1,NRMSEP)
                book1.save("report.xls")
                RMSEP.append(RMSEP1)
        RMSEPpred=[]
        StdDevRMSEP=[]
        for i in range(len(RMSEP[0])):
                RMSEPpred1=[]
                for j in range(len(RMSEP)):
                        RMSEPpred2=RMSEP[j][i]
                        RMSEPpred1.append(RMSEPpred2)
                RMSEPpred.append(mean(RMSEPpred1))
                StdDevRMSEP1=round(stdev(RMSEPpred1),2)
                StdDevRMSEP.append(StdDevRMSEP1)
        book= xlrd.open_workbook("report.xls")
        book1= copy(book)
        sheet1 = book1.get_sheet("Model")
        for i in range(len(RMSEPpred)):
                if round(RMSEPpred[i]) > 0.01:

        sheet1.write((3+folds)+i*(6+folds),ModelNumber,round(RMSEPpred[i],2)
)
                else:
                        sheet1.write((3+folds)+i*(6+folds),ModelNumber, "<0.00")

        sheet1.write((4+folds)+i*(6+folds),ModelNumber,StdDevRMSEP[i])
        book1.save("report.xls")
        return RMSEPpred
pass
def
PCR(QCTableColums,PTableColums,CompleteDataSet,trainingIndex,validatio
nIndex,folds,PCANoComponents):
        RMSEP=[]
        ModelNumber=7
        ModelName="PCR"
        for i in range(len(trainingIndex)):
```

```
        Output=XYCreation(QCTableColums,PTableColums,CompleteDataSet,t
rainingIndex[i],validationIndex[i])
                X=Output[0]
                XValidation=Output[1]
                Y=Output[2]
                YRange1=Output[3]
                if PCANoComponents > len(trainingIndex[0]):
                        PCANoComponents = len(trainingIndex[0])
                pca = PCA(n_components=PCANoComponents)
                XPCR=pca.fit_transform(X)
                mlr1=LinearRegression().fit(XPCR, Y)
                XPCRValidation=pca.fit_transform(XValidation)
                Y_pred=mlr1.predict(XPCRValidation)

        RMSEP1=RMSEPReport(QCTableColums,PTableColums,CompleteDat
aSet,trainingIndex[i],validationIndex[i],ModelNumber,ModelName,Y_pred,i,YRa
nge1,folds)
                RMSEP.append(RMSEP1)
        RMSEPpred=[]
        StdDevRMSEP=[]
        for i in range(len(RMSEP[0])):
                RMSEPpred1=[]
                for j in range(len(RMSEP)):
                        RMSEPpred2=RMSEP[j][i]
                        RMSEPpred1.append(RMSEPpred2)
                RMSEPpred.append(mean(RMSEPpred1))
                StdDevRMSEP1=round(stdev(RMSEPpred1),2)
                StdDevRMSEP.append(StdDevRMSEP1)
        book= xlrd.open_workbook("report.xls")
        book1= copy(book)
        sheet1 = book1.get_sheet("Model")
        for i in range(len(RMSEPpred)):
                if round(RMSEPpred[i]) > 0.01:

        sheet1.write((3+folds)+i*(6+folds),ModelNumber,round(RMSEPpred[i],2)
)
                else:
                        sheet1.write((3+folds)+i*(6+folds),ModelNumber, "<0.00")

        sheet1.write((4+folds)+i*(6+folds),ModelNumber,StdDevRMSEP[i])
        book1.save("report.xls")
        return RMSEPpred
pass
```

```
def
RTboosted(QCTableColums,PTableColums,CompleteDataSet,trainingIndex,va
lidationIndex,folds,RTbDepth,RtbEstimators):
        RMSEP=[]
        ModelNumber=9
        ModelName= "RT boosted"
        for i in range(len(trainingIndex)):
                X=[]
                XValidation=[]
                Y=[]
                for j in range(len(trainingIndex[i])):
                        Xi=[]
                        Xi.append(float((CompleteDataSet[trainingIndex[i][j]][2])))
        # Linespeed
                        for k in range(len(PTableColums)-2):
                                variablePlace= 4 + len(QCTableColums)-2

        Xi.append(float(CompleteDataSet[trainingIndex[i][j]][variablePlace+k]))
                        X.append(Xi)
                for j in range(len(validationIndex[i])):
                        XiValidation=[]

        XiValidation.append(CompleteDataSet[validationIndex[i][j]][2])        #
Linespeed
                        for k in range(len(PTableColums)-2):
                                variablePlace= 4 + len(QCTableColums)-2

        XiValidation.append(float(CompleteDataSet[validationIndex[i][j]][variable
Place+k]))
                        XValidation.append(XiValidation)
                Y_pred=[]
                YRange=[]
                for j in range(len(QCTableColums)-2):
                        Y=[]
                        for k in range(len(trainingIndex[i])):

        Y.append(float(CompleteDataSet[trainingIndex[i][k]][4+j]))
                        YRange.append(max(Y)-min(Y))
                        rng = np.random.RandomState(1)
                        rtb =
AdaBoostRegressor(DecisionTreeRegressor(max_depth=RTbDepth),n_estima
tors=RtbEstimators, random_state=rng)
                        rtb.fit(X, Y)
                        Y_pred1=rtb.predict(XValidation)
                        Y_pred.append(Y_pred1)
```

```
                RMSEP1=[]
                for j in range(len(QCTableColums)-2):
                        RMSEP2=[]
                        for k in range(len(validationIndex[i])):
                                RMSEP3=math.sqrt((Y_pred[j][k]-
float(CompleteDataSet[validationIndex[i][k]][4+j]))**2)
                                RMSEP2.append(RMSEP3)
                        RMSEP1.append(mean(RMSEP2))
                book= xlrd.open_workbook("report.xls")
                book1= copy(book)
                sheet1 = book1.get_sheet("Model")
                for j in range(len(RMSEP1)):
                        sheet1.write(1+j*(6+folds),ModelNumber,ModelName)
                        sheet1.write(2+j*(6+folds),ModelNumber,"RMSEP")
                        sheet1.write(2+j*(6+folds),ModelNumber+1,"NRMSEP")

        sheet1.write(3+j*(6+folds)+i,ModelNumber,round(RMSEP1[j],2))
                        try:
                                NRMSEP=round(RMSEP1[j]/YRange[j] *100,2)
                        except:
                                NRMSEP = "NA"
                        sheet1.write(3+(6+folds)*j+i,ModelNumber+1,NRMSEP)
                book1.save("report.xls")
                RMSEP.append(RMSEP1)
        RMSEPpred=[]
        StdDevRMSEP=[]
        for i in range(len(RMSEP[0])):
                RMSEPpred1=[]
                for j in range(len(RMSEP)):
                        RMSEPpred2=RMSEP[j][i]
                        RMSEPpred1.append(RMSEPpred2)
                RMSEPpred.append(mean(RMSEPpred1))
                StdDevRMSEP1=round(stdev(RMSEPpred1),2)
                StdDevRMSEP.append(StdDevRMSEP1)
        book= xlrd.open_workbook("report.xls")
        book1= copy(book)
        sheet1 = book1.get_sheet("Model")
        for i in range(len(RMSEPpred)):
                if round(RMSEPpred[i]) > 0.01:

        sheet1.write((3+folds)+i*(6+folds),ModelNumber,round(RMSEPpred[i],2)
)
                else:
                        sheet1.write((3+folds)+i*(6+folds),ModelNumber, "<0.00")
```

```
        sheet1.write((4+folds)+i*(6+folds),ModelNumber,StdDevRMSEP[i])
        book1.save("report.xls")
        return RMSEPpred
pass
def
kNN(QCTableColums,PTableColums,CompleteDataSet,trainingIndex,validatio
nIndex,folds,kNNNeigbors):
        RMSEP=[]
        ModelNumber=11
        ModelName= "kNN"
        for i in range(len(trainingIndex)):
                X=[]
                XValidation=[]
                Y=[]
                for j in range(len(trainingIndex[i])):
                        Xi=[]
                        Xi.append(float((CompleteDataSet[trainingIndex[i][j]][2])))
        # Linespeed
                        for k in range(len(PTableColums)-2):
                                variablePlace= 4 + len(QCTableColums)-2

        Xi.append(float(CompleteDataSet[trainingIndex[i][j]][variablePlace+k]))
                        X.append(Xi)
                for j in range(len(validationIndex[i])):
                        XiValidation=[]

        XiValidation.append(CompleteDataSet[validationIndex[i][j]][2])        #
Linespeed
                        for k in range(len(PTableColums)-2):
                                variablePlace= 4 + len(QCTableColums)-2

        XiValidation.append(float(CompleteDataSet[validationIndex[i][j]][variable
Place+k]))
                        XValidation.append(XiValidation)
                Y_pred=[]
                YRange=[]
                for j in range(len(QCTableColums)-2):
                        Y=[]
                        for k in range(len(trainingIndex[i])):

        Y.append(float(CompleteDataSet[trainingIndex[i][k]][4+j]))
                        YRange.append(max(Y)-min(Y))
                        if len(trainingIndex[0]) < kNNNeigbors:
                                kNNNeigbors = len(trainingIndex[0])-1
```

```
                        neigh =
KNeighborsRegressor(n_neighbors=kNNNeigbors)
                        neigh.fit(X, Y)
                        Y_pred1=neigh.predict(XValidation)
                        Y_pred.append(Y_pred1)
                RMSEP1=[]
                for j in range(len(QCTableColums)-2):
                        RMSEP2=[]
                        for k in range(len(validationIndex[i])):
                                RMSEP3=math.sqrt((Y_pred[j][k]-
float(CompleteDataSet[validationIndex[i][k]][4+j]))**2)
                                RMSEP2.append(RMSEP3)
                        RMSEP1.append(mean(RMSEP2))
                book= xlrd.open_workbook("report.xls")
                book1= copy(book)
                sheet1 = book1.get_sheet("Model")
                for j in range(len(RMSEP1)):
                        sheet1.write(1+j*(6+folds),ModelNumber,ModelName)
                        sheet1.write(2+j*(6+folds),ModelNumber,"RMSEP")
                        sheet1.write(2+j*(6+folds),ModelNumber+1,"NRMSEP")

        sheet1.write(3+j*(6+folds)+i,ModelNumber,round(RMSEP1[j],2))
                        try:
                                NRMSEP=round(RMSEP1[j]/YRange[j] *100,2)
                        except:
                                NRMSEP = "NA"
                        sheet1.write(3+j*(6+folds)+i,ModelNumber+1,NRMSEP)
                book1.save("report.xls")
                RMSEP.append(RMSEP1)
        RMSEPpred=[]
        StdDevRMSEP=[]
        for i in range(len(RMSEP[0])):
                RMSEPpred1=[]
                for j in range(len(RMSEP)):
                        RMSEPpred2=RMSEP[j][i]
                        RMSEPpred1.append(RMSEPpred2)
                RMSEPpred.append(mean(RMSEPpred1))
                StdDevRMSEP1=round(stdev(RMSEPpred1),2)
                StdDevRMSEP.append(StdDevRMSEP1)
        book= xlrd.open_workbook("report.xls")
        book1= copy(book)
        sheet1 = book1.get_sheet("Model")
        for i in range(len(RMSEPpred)):
                if round(RMSEPpred[i]) > 0.01:
```

```
        sheet1.write((3+folds)+i*(6+folds),ModelNumber,round(RMSEPpred[i],2)
)
                else:
                        sheet1.write((3+folds)+i*(6+folds),ModelNumber, "<0.00")

        sheet1.write((4+folds)+i*(6+folds),ModelNumber,StdDevRMSEP[i])
        book1.save("report.xls")
        return RMSEPpred
pass
def
LASSO(QCTableColums,PTableColums,CompleteDataSet,trainingIndex,valida
tionIndex,folds,LASSOAlpha):
        RMSEP=[]
        ModelName="LASSO"
        ModelNumber=13
        for i in range(len(trainingIndex)):
                X=[]
                XValidation=[]
                Y=[]
                for j in range(len(trainingIndex[i])):
                        Xi=[]
                        Xi.append(float((CompleteDataSet[trainingIndex[i][j]][2])))
        # Linespeed
                        for k in range(len(PTableColums)-2):
                                variablePlace= 4 + len(QCTableColums)-2

        Xi.append(float(CompleteDataSet[trainingIndex[i][j]][variablePlace+k]))
                        X.append(Xi)
                for j in range(len(validationIndex[i])):
                        XiValidation=[]

        XiValidation.append(CompleteDataSet[validationIndex[i][j]][2])        #
Linespeed
                        for k in range(len(PTableColums)-2):
                                variablePlace= 4 + len(QCTableColums)-2

        XiValidation.append(float(CompleteDataSet[validationIndex[i][j]][variable
Place+k]))
                        XValidation.append(XiValidation)
                Y_pred=[]
                YRange=[]
                for j in range(len(QCTableColums)-2):
                        Y=[]
                        for k in range(len(trainingIndex[i])):
```

```
        Y.append(float(CompleteDataSet[trainingIndex[i][k]][4+j]))
                        YRange.append(max(Y)-min(Y))
                        lasso = Lasso(alpha=LASSOAlpha)
                        lasso.fit(X,Y)
                        Y_pred1=lasso.predict(XValidation)
                        Y_pred.append(Y_pred1)
                RMSEP1=[]
                for j in range(len(QCTableColums)-2):
                        RMSEP2=[]
                        for k in range(len(validationIndex[i])):
                                RMSEP3=math.sqrt((Y_pred[j][k]-
float(CompleteDataSet[validationIndex[i][k]][4+j]))**2)
                                RMSEP2.append(RMSEP3)
                        RMSEP1.append(mean(RMSEP2))
                book= xlrd.open_workbook("report.xls")
                book1= copy(book)
                sheet1 = book1.get_sheet("Model")
                for j in range(len(RMSEP1)):
                        sheet1.write(1+j*(6+folds),ModelNumber,ModelName)
                        sheet1.write(2+j*(6+folds),ModelNumber,"RMSEP")
                        sheet1.write(2+j*(6+folds),ModelNumber+1,"NRMSEP")

        sheet1.write(3+j*(6+folds)+i,ModelNumber,round(RMSEP1[j],2))
                        try:
                                NRMSEP=round(RMSEP1[j]/YRange[j] *100,2)
                        except:
                                NRMSEP = "NA"
                        sheet1.write(3+j*(6+folds)+i,ModelNumber+1,NRMSEP)
                book1.save("report.xls")
                RMSEP.append(RMSEP1)
        RMSEPpred=[]
        StdDevRMSEP=[]
        for i in range(len(RMSEP[0])):
                RMSEPpred1=[]
                for j in range(len(RMSEP)):
                        RMSEPpred2=RMSEP[j][i]
                        RMSEPpred1.append(RMSEPpred2)
                RMSEPpred.append(mean(RMSEPpred1))
                StdDevRMSEP1=round(stdev(RMSEPpred1),2)
                StdDevRMSEP.append(StdDevRMSEP1)
        book= xlrd.open_workbook("report.xls")
        book1= copy(book)
        sheet1 = book1.get_sheet("Model")
        for i in range(len(RMSEPpred)):
```

```
                if round(RMSEPpred[i]) > 0.01:

        sheet1.write((3+folds)+i*(6+folds),ModelNumber,round(RMSEPpred[i],2)
)
                else:
                        sheet1.write((3+folds)+i*(6+folds),ModelNumber, "<0.00")

        sheet1.write((4+folds)+i*(6+folds),ModelNumber,StdDevRMSEP[i])
        book1.save("report.xls")
        return RMSEPpred
pass
def
RidgeRegression(QCTableColums,PTableColums,CompleteDataSet,trainingIn
dex,validationIndex,folds,RidgeAlpha):
        RMSEP=[]
        ModelName="Ridge"
        ModelNumber=15
        for i in range(len(trainingIndex)):
                X=[]
                XValidation=[]
                Y=[]
                for j in range(len(trainingIndex[i])):
                        Xi=[]
                        Xi.append(float((CompleteDataSet[trainingIndex[i][j]][2])))
        # Linespeed
                        for k in range(len(PTableColums)-2):
                                variablePlace= 4 + len(QCTableColums)-2

        Xi.append(float(CompleteDataSet[trainingIndex[i][j]][variablePlace+k]))
                        X.append(Xi)
                for j in range(len(validationIndex[i])):
                        XiValidation=[]

        XiValidation.append(CompleteDataSet[validationIndex[i][j]][2])        #
Linespeed
                        for k in range(len(PTableColums)-2):
                                variablePlace= 4 + len(QCTableColums)-2

        XiValidation.append(float(CompleteDataSet[validationIndex[i][j]][variable
Place+k]))
                        XValidation.append(XiValidation)
                Y_pred=[]
                YRange=[]
                for j in range(len(QCTableColums)-2):
                        Y=[]
```

```
                    for k in range(len(trainingIndex[i])):

        Y.append(float(CompleteDataSet[trainingIndex[i][k]][4+j]))
                    YRange.append(max(Y)-min(Y))
                    ridge = Ridge(alpha=RidgeAlpha)
                    ridge.fit(X,Y)
                    Y_pred1=ridge.predict(XValidation)
                    Y_pred.append(Y_pred1)
            RMSEP1=[]
            for j in range(len(QCTableColums)-2):
                    RMSEP2=[]
                    for k in range(len(validationIndex[i])):
                            RMSEP3=math.sqrt((Y_pred[j][k]-
float(CompleteDataSet[validationIndex[i][k]][4+j]))**2)
                            RMSEP2.append(RMSEP3)
                    RMSEP1.append(mean(RMSEP2))
            book= xlrd.open_workbook("report.xls")
            book1= copy(book)
            sheet1 = book1.get_sheet("Model")
            for j in range(len(RMSEP1)):
                    sheet1.write(1+j*(6+folds),ModelNumber,ModelName)
                    sheet1.write(2+j*(6+folds),ModelNumber,"RMSEP")
                    sheet1.write(2+j*(6+folds),ModelNumber+1,"NRMSEP")

        sheet1.write(3+j*(6+folds)+i,ModelNumber,round(RMSEP1[j],2))
                    try:
                            NRMSEP=round(RMSEP1[j]/YRange[j] *100,2)
                    except:
                            NRMSEP = "NA"
                    sheet1.write(3+j*(6+folds)+i,ModelNumber+1,NRMSEP)
            book1.save("report.xls")
            RMSEP.append(RMSEP1)
        RMSEPpred=[]
        StdDevRMSEP=[]
        for i in range(len(RMSEP[0])):
            RMSEPpred1=[]
            for j in range(len(RMSEP)):
                    RMSEPpred2=RMSEP[j][i]
                    RMSEPpred1.append(RMSEPpred2)
            RMSEPpred.append(mean(RMSEPpred1))
            StdDevRMSEP1=round(stdev(RMSEPpred1),2)
            StdDevRMSEP.append(StdDevRMSEP1)
        book= xlrd.open_workbook("report.xls")
        book1= copy(book)
        sheet1 = book1.get_sheet("Model")
```

```
        for i in range(len(RMSEPpred)):
                if round(RMSEPpred[i]) > 0.01:

        sheet1.write((3+folds)+i*(6+folds),ModelNumber,round(RMSEPpred[i],2)
)
                else:
                        sheet1.write((3+folds)+i*(6+folds),ModelNumber, "<0.00")

        sheet1.write((4+folds)+i*(6+folds),ModelNumber,StdDevRMSEP[i])
        book1.save("report.xls")
        return RMSEPpred
pass
def
RMSEP(QCTableColums,PTableColums,CompleteDataSet,trainingIndex,valid
ationIndex,folds,NNAlpha,HiddenLayers,PCANoComponents,RTbDepth,RtbEs
timators,kNNNeigbors,LASSOAlpha,RidgeAlpha):
        ModelMethods=["MLR","PLS","NN","PCR","RTboosted","kNN","LASSO",
"RidgeRegression"]
        RMSEP=[]
        #print("MLR")
        x=MLR(QCTableColums,PTableColums,CompleteDataSet,trainingIndex
,validationIndex,folds)
        RMSEP.append(x)
        #print("PLS")
        x=PLS(QCTableColums,PTableColums,CompleteDataSet,trainingIndex,
validationIndex,folds)
        RMSEP.append(x)
        #print("NN")
        x=NN(QCTableColums,PTableColums,CompleteDataSet,trainingIndex,v
alidationIndex,folds,NNAlpha,HiddenLayers)
        RMSEP.append(x)
        #print("PCR")
        x=PCR(QCTableColums,PTableColums,CompleteDataSet,trainingIndex
,validationIndex,folds,PCANoComponents)
        RMSEP.append(x)
        #print("RTboosted")
        x=RTboosted(QCTableColums,PTableColums,CompleteDataSet,trainin
gIndex,validationIndex,folds,RTbDepth,RtbEstimators)
        RMSEP.append(x)
        #print("kNN")
        x=kNN(QCTableColums,PTableColums,CompleteDataSet,trainingIndex,
validationIndex,folds,kNNNeigbors)
        RMSEP.append(x)
        #print("LASSO")
```

```
        x=LASSO(QCTableColums,PTableColums,CompleteDataSet,trainingIn
dex,validationIndex,folds,LASSOAlpha)
        RMSEP.append(x)
        #print("Ridge")
        x=RidgeRegression(QCTableColums,PTableColums,CompleteDataSet,t
rainingIndex,validationIndex,folds,RidgeAlpha)
        RMSEP.append(x)

        ModelSelection=[]
        for i in range(len(RMSEP[0])):
                ModelSelection1=[]
                for j in range(len(RMSEP)):
                        ModelSelection1.append(RMSEP[j][i])
                ModelSelection.append(min(ModelSelection1))
        count=0
        Model=[[]for i in range(len(RMSEP[0]))]
        for i in range(len(RMSEP[0])):
                for j in range(len(RMSEP)):
                        if RMSEP[j][i] == ModelSelection[i]:
                                Model[i]=ModelMethods[j]
                                count=+1
        return Model
pass
```

ModelMethodCalculation.py

```
import mysql.connector
import random
import math
from statistics import mean
from sklearn.linear_model import LinearRegression
from sklearn.cross_decomposition import PLSRegression
from sklearn.neural_network import MLPRegressor
from sklearn.decomposition import PCA
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge

#  Connect ot the Database
mydb = mysql.connector.connect(
        host ="localhost",
```

```
        user = "root",
        passwd = "Root",
        database = "program")

#       Object that communicates with the mySQL Database
mycursor = mydb.cursor()

def
ModelMethodCreation(SelectedVariables,Method,NNAlpha,HiddenLayers,PCA
NoComponents,RTbDepth,RtbEstimators,kNNNeigbors,LASSOAlpha,RidgeAlp
ha):

        #       Find all Colums in the QC Table
        QCTableColums= []
        sqlFormula2= "SHOW COLUMNS FROM qualitycontrol"
        mycursor.execute(sqlFormula2)
        myresult = mycursor.fetchall()
        for row in myresult:
                QCTableColums.append(row[0])
        mydb.commit()

        CompleteDataSet2=[]
        VariableString1= []
        VariableString3= []
        for i in range(len(Method)):
                Variables= SelectedVariables[i]
                VariableString = ""
                for x in range(len(Variables)-1):
                  VariableString = VariableString +"`"+ Variables[x] +"`"+ " IS NOT
NULL AND "
                VariableString = VariableString+"`" + Variables[len(Variables)-
1]+"`"
                VariableString2 = ""
                VariableString4 = ""
                for x in range(len(Variables)-1):
                        VariableString2 = VariableString2 +"`"+ Variables[x] +"`"+
", "
                for x in range(len(QCTableColums)-2):
                        VariableString4 = VariableString4 +"`"+
QCTableColums[x+2] +"`"+ ", "
                VariableString2 = "TimeStamp, InputNo, Linespeed, Assessed,
"+VariableString4+VariableString2+"`" + Variables[len(Variables)-1]+"`"
```

```
                sqlFormula1= "SELECT {} FROM program.fuseddatameasured
WHERE {} IS NOT NULL AND {} IS NOT
NULL".format(VariableString2,QCTableColums[2+i],VariableString)
                mycursor.execute(sqlFormula1)
                myresult = mycursor.fetchall()
                CompleteDataSet1=[]
                for row in myresult:
                        Dataset= []
                        for i in range(len(row)):
                                Dataset.append(row[i])
                        CompleteDataSet1.append(Dataset)
                mydb.commit()
                CompleteDataSet2.append(CompleteDataSet1)

        X3=[]
        for k in range(len(Method)):
                CompleteDataSet=CompleteDataSet2[k]
                X2=[]
                for i in range(len(CompleteDataSet)): #Zeilen
                        X1=[]
                        X1.append(float(CompleteDataSet[i][2]))
                        for j in range(len(CompleteDataSet[i])-4-
(len(QCTableColums)-2)): #Spalten
                                positionno= 4+(len(QCTableColums)-2)
                                X1.append(float(CompleteDataSet[i][positionno+j]))
                        X2.append(X1)
                X3.append(X2)

        CalculatedModel=[]
        for i in range(len(Method)):
                Y=[]
                CompleteDataSet=CompleteDataSet2[i]
                X=X3[i]
                for j in range(len(CompleteDataSet)): #Zeilen
                        position=4
                        Y1=[float(CompleteDataSet[j][4+i])]
                        Y.append(Y1)

        #ModelMethods=["MLR","PLS","NN","PCR","RTboosted","kNN","LASSO
","RidgeRegression"]
                if Method[i] == "MLR":
                        mlr1= MLR(X,Y)
                        #print("MLR")
                        CalculatedModel.append(mlr1)
                elif Method[i] == "PLS":
```

```
                        pls1=PLS(X,Y)
                        #print("PLS")
                        CalculatedModel.append(pls1)
                elif Method[i] == "NN":
                        nn=NN(X,Y,NNAlpha,HiddenLayers)
                        #print("NN")
                        CalculatedModel.append(nn)
                elif Method[i] == "PCR":
                        pca=PCR(X,Y,PCANoComponents)
                        #print("PCR")
                        CalculatedModel.append(pca)
                elif Method[i] == "RTboosted":
                        rtb=RTboosted(X,Y,RTbDepth,RtbEstimators)
                        #print("RTboosted")
                        CalculatedModel.append(rtb)
                elif Method[i] == "kNN":
                        neigh=kNN(X,Y,kNNNeigbors)
                        #print("kNN")
                        CalculatedModel.append(neigh)
                elif Method[i] == "LASSO":
                        lasso=LASSO(X,Y,LASSOAlpha)
                        #print("LASSO")
                        CalculatedModel.append(lasso)
                elif Method[i] == "RidgeRegression":
                        ridge=RidgeRegression(X,Y,RidgeAlpha)
                        #print("RidgeRegression")
                        CalculatedModel.append(ridge)
                else:
                        print("Calcualtion = something is totally wrong!")
        return CalculatedModel
pass

def MLR(X,Y):
        mlr1=LinearRegression().fit(X, Y)
        return mlr1
pass
def PLS(X,Y):
        pls1 = PLSRegression(copy=True, max_iter=500, scale=True, tol=1e-
06)
        pls1.fit(X, Y)
        return pls1
pass
def NN(X,Y,NNAlpha,HiddenLayers):
        nn = MLPRegressor(solver='lbfgs',
alpha=NNAlpha,hidden_layer_sizes=HiddenLayers, random_state=1)
```

```
        nn.fit(X, np.ravel(Y,order='C'))
        return nn
pass
def PCR(X,Y,PCANoComponents):
        if PCANoComponents > len(Y):
                PCANoComponents = len(Y)
        pca=PCA(n_components=PCANoComponents)
        pca.fit(X)
        X2=pca.transform(X)
        mlr=LinearRegression().fit(X2, Y)
        Model=[]
        Model.append(pca)
        Model.append(mlr)
        return Model
pass
def RTboosted(X,Y,RTbDepth,RtbEstimators):
        rtb =
AdaBoostRegressor(DecisionTreeRegressor(max_depth=RTbDepth),n_estima
tors=RtbEstimators, random_state=np.random.RandomState(1))
        rtb.fit(X, np.ravel(Y,order='C'))
        return rtb
pass
def kNN(X,Y,kNNNeigbors):
        if len(Y) < kNNNeigbors:
                kNNNeigbors = len(Y)-1
        neigh = KNeighborsRegressor(n_neighbors=kNNNeigbors)
        neigh.fit(X, np.ravel(Y,order='C'))
        return neigh
pass
def LASSO(X,Y,LASSOAlpha):
        lasso = Lasso(alpha=LASSOAlpha)
        lasso.fit(X,Y)
        return lasso
pass
def RidgeRegression(X,Y,RidgeAlpha):
        ridge = Ridge(alpha=RidgeAlpha)
        ridge.fit(X,Y)
        return ridge
pass
```

ModelImputation.py

```
import mysql.connector
import random
import math
```

```
from statistics import mean
from sklearn.linear_model import LinearRegression
from sklearn.cross_decomposition import PLSRegression
from sklearn.neural_network import MLPClassifier
from sklearn.decomposition import PCA
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge


#  Connect ot the Database
mydb = mysql.connector.connect(
        host ="localhost",
        user = "root",
        passwd = "Root",
        database = "program")

#       Object that communicates with the mySQL Database
mycursor = mydb.cursor()

def ModelPrediction(SelectedVariables,CalculatedModel,Model):
        #       Find all Colums in the QC Table
        QCTableColums= []
        sqlFormula1= "SHOW COLUMNS FROM qualitycontrol"
        mycursor.execute(sqlFormula1)
        myresult = mycursor.fetchall()
        for row in myresult:
                QCTableColums.append(row[0])
        mydb.commit()
        #       put Variables in a string
        VariableString1= []
        VariableString3= []
        for i in range(len(CalculatedModel)):
                Variables=SelectedVariables[i]
                VariableString = ""
                for x in range(len(Variables)-1):
                  VariableString = VariableString +"`"+ Variables[x] +"`"+ " IS NOT
NULL AND "
                VariableString = VariableString+"`" + Variables[len(Variables)-
1]+"`"
                NoRowsPredicted = 0
                VariableString1.append(VariableString)
```

```
                VariableString2 = ""
                VariableString4 = ""
                for x in range(len(Variables)-1):
                        VariableString2 = VariableString2 +"`"+ Variables[x] +"`"+
", "
                for x in range(len(QCTableColums)-2):
                        VariableString4 = VariableString4 +"`"+
QCTableColums[x+2] +"`"+ ", "
                VariableString2 = "TimeStamp, Assessed, Linespeed,
"+VariableString4+VariableString2+"`" + Variables[len(Variables)-1]+"`"
                VariableString3.append(VariableString2)

        for i in range(len(CalculatedModel)):
                DataInput=[]
                VariableString=VariableString1[i]
                VariableString2=VariableString3[i]
                sqlFormula2="SELECT {} FROM program.fuseddataprediction
WHERE {} IS NULL AND {} IS NOT
NULL".format(VariableString2,QCTableColums[2+i],VariableString)
                mycursor.execute(sqlFormula2)
                myresult = mycursor.fetchall()
                for row in myresult:
                        DataInput.append(row)
                try:
                        for j in range(len(DataInput)):                 #row
                                time=DataInput[j][0]
                                X=[]
                                X.append(float(DataInput[j][2]))
                                for k in range(len(SelectedVariables[i])):          #
Variables
                                        position=len(QCTableColums)-2
                                        X.append(float(DataInput[j][position+3+k]))
                                X1=[]
                                X1.append(X)
                                if Model[i] == "PCR":
                                        try:

        X2=CalculatedModel[i][0].transform(X1)

        Xpred=CalculatedModel[i][1].predict(X2)
                                                Xpred=round(float(Xpred),2)
                                        except:
                                                Xpred = "NULL"
```

```
                                        sqlFormula3="UPDATE
`program`.`fuseddataprediction` SET `{}` = '{}' WHERE (`TimeStamp` =
'{}')".format(QCTableColums[2+i],Xpred,time)
                                        mycursor.execute(sqlFormula3)
                                        mydb.commit()
                                else:
                                        try:
                                                Xpred=CalculatedModel[i].predict(X1)
                                                Xpred=round(float(Xpred),2)
                                        except:
                                                Xpred = "NULL"
                                        sqlFormula3="UPDATE
`program`.`fuseddataprediction` SET `{}` = '{}' WHERE (`TimeStamp` =
'{}')".format(QCTableColums[2+i],Xpred,time)
                                        mycursor.execute(sqlFormula3)
                                        mydb.commit()
                        except:
                                print("No predicted Values")

        NoRowsPredicted =0
        return NoRowsPredicted
pass
```

ResearchReport.py

```
import mysql.connector
import random
import math
from statistics import mean, median, stdev
import xlwt
import xlrd
from xlutils.copy import copy
import time
import datetime
import os

#  Connect ot the Database
mydb = mysql.connector.connect(
        host ="localhost",
        user = "root",
        passwd = "Root",
        database = "program")

#       Object that communicates with the mySQL Database
mycursor = mydb.cursor()
```

```
def
research(number,name,TimeBegin,TimeAlignment,TimeQAMinMax,TimeGrubs
,TimeImputation,TimePrediction,Method,Model,DeletedRowsM,DeletedRowsP,
OutlierRowsM,OutlierRowsP,NoRowsPredicted):
        #       Find all Colums in the QC Table
        QCTableColums= []
        sqlFormula2= "SHOW COLUMNS FROM program.qualitycontrol"
        mycursor.execute(sqlFormula2)
        myresult = mycursor.fetchall()
        for row in myresult:
                QCTableColums.append(row[0])
        mydb.commit()
        # create excel sheet
        book= xlrd.open_workbook("research.xls")
        book1= copy(book)
        sheet_names = ["QC1","QC2","QC3","Method","Model"]
        for i in range(len(QCTableColums)-2):
                sheet1 = book1.get_sheet(sheet_names[i])
                sheet1.write(0,number,name)
                time1 = str(TimePrediction-TimeBegin)
                sheet1.write(1,number,time1)                    #total time
                time2 = str(TimeAlignment-TimeBegin)
                sheet1.write(2,number,time2)                    # time alignment
                time3 = str(TimeQAMinMax-TimeAlignment)
                sheet1.write(3,number,time3)                    # time min max
                time4 = str(TimeGrubs-TimeQAMinMax)
                sheet1.write(4,number,time4)                    # time Grubs
                time5 = str(TimeImputation-TimeGrubs)
                sheet1.write(5,number,time5)                    # time imputation
                time6 = str(TimePrediction-TimeImputation)
                sheet1.write(6,number,time6)                    # time prediction
                currentColum= QCTableColums[i+2]
                sqlFormula1="Select {} FROM
program.fuseddataprediction".format(currentColum)
                mycursor.execute(sqlFormula1)
                myresult = mycursor.fetchall()
                QC_predVlaues=[]
                for row in myresult:
                        QC_predVlaues.append(row[0])
                for j in range(len(QC_predVlaues)):
                        sheet1.write(j+7,number,QC_predVlaues[j])

        sheet1 = book1.get_sheet("Method")
        sheet1.write(0,number,name)
```

```
for i in range(len(Method)):
        sheet1.write(i,number,Method[i])

sheet1 = book1.get_sheet("Model")
sheet1.write(0,number,name)
for i in range(len(Model)):
        sheet1.write(i,number,Model[i])

sheet1 = book1.get_sheet("Values")
sheet1.write(number+1,0,number)
sheet1.write(number+1,1,name)
sheet1.write(number+1,2,DeletedRowsM)
sheet1.write(number+1,3,DeletedRowsP)
sheet1.write(number+1,4,OutlierRowsM)
sheet1.write(number+1,5,OutlierRowsP)
sheet1.write(number+1,6,NoRowsPredicted)


book1.save("research.xls")



book= xlrd.open_workbook("report.xls")
book1= copy(book)
sheet1 = book1.add_sheet("Sensordata")
sqlFormula1="SELECT * FROM program.sensordata"
mycursor.execute(sqlFormula1)
myresult = mycursor.fetchall()
for i in range(len(myresult)):                          #Row
        for j in range(len(myresult[i])):
#Colum
                try:
                        sheet1.write(i,j,myresult[i][j])
                except:
                        print(myresult[i][j])
                        value= str(myresult[i][j])
                        sheet1.write(i,j,value)

sheet2 = book1.add_sheet("Qualitycontrol")
sqlFormula2="SELECT * FROM program.qualitycontrol"
mycursor.execute(sqlFormula2)
myresult = mycursor.fetchall()
for i in range(len(myresult)):                          #Row
        for j in range(len(myresult[i])):
#Colum
```

```
                    try:
                            sheet2.write(i,j,myresult[i][j])
                    except:
                            value= str(myresult[i][j])
                            sheet2.write(i,j,value)

sheet3 = book1.add_sheet("Sensordistance")
sqlFormula3="SELECT * FROM program.sensordistance"
mycursor.execute(sqlFormula3)
myresult = mycursor.fetchall()
for i in range(len(myresult)):                                  #Row
        for j in range(len(myresult[i])):
#Colum
                    try:
                            sheet3.write(i,j,myresult[i][j])
                    except:
                            value= str(myresult[i][j])
                            sheet3.write(i,j,value)

sheet4 = book1.add_sheet("fuseddatameasured")
sqlFormula4= "SELECT * FROM program.fuseddatameasured"
mycursor.execute(sqlFormula4)
myresult = mycursor.fetchall()
for i in range(len(myresult)):                                  #Row
        for j in range(len(myresult[i])):
#Colum
                    try:
                            sheet4.write(i,j,myresult[i][j])
                    except:
                            value= str(myresult[i][j])
                            sheet4.write(i,j,value)

sheet5 = book1.add_sheet("fuseddataprediction")
sqlFormula5="SELECT * FROM program.fuseddataprediction"
mycursor.execute(sqlFormula5)
myresult = mycursor.fetchall()
for i in range(len(myresult)):                                  #Row
        for j in range(len(myresult[i])):
#Colum
                    try:
                            sheet5.write(i,j,myresult[i][j])
                    except:
                            value= str(myresult[i][j])
                            sheet5.write(i,j,value)
```

```
        book1.save("{}.xls".format(name))
        os.remove("report.xls")

pass
```

VITA

David Christoph Juriga was born in Hallein, Austria, where he accomplished his A-levels and an apprenticeship to a joiner and interior designer. He attended the University of Applied Sciences in Salzburg, Austria where he earned a B.S. in forest products technology.

During his undergraduate program David dealt with the “creation of an integrated management handbook with main focus on quality management”. Prior to the graduate program at UT he worked for the Austrian company Dana, one of the largest door producers in Austria and leader on the Austrian market. He currently is studying under Dr. Timothy Young are the Center for Renewable Carbon and his M.S. research focuses on improving the data quality of manufacturing data.

He plans to graduate from the Salzburg University of Applied Sciences with a graduate engineer in “Forest Products Technology and Management” in September 2019. Also, he plans to graduate from the University of Tennessee with a “Master of Science in Wood Science, Technology and Biomaterials” in December 2019.